Advanced OpenGL Optimizations

Session 209

Advanced OpenGL Optimizations

John Stauffer
OpenGL Engineering Manager
stauffer@apple.com

What You'll Learn

- Basic things to avoid
- Optimizing texture uploads
- Optimizing vertex data throughput
- Optimizing for one shot images
- Optimizing pixel copy operations
- Using threads with OpenGL
- Using the ***OpenGL Profiler***

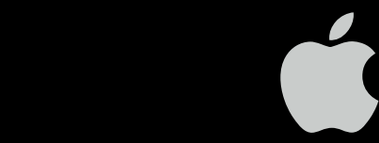
Goals of Optimizing OpenGL

- Maximize performance
  - Effective use of both CPU and GPU

- Minimize CPU burden
  - Offload work to GPU

Key Concepts to Keep In Mind

- Eliminate CPU copies of data
  - Have GPU read pixel and vertex data directly
- Cache static data in VRAM
- Maximize asynchronous behavior between CPU and GPU
- Use Threads



Basic Things to Avoid

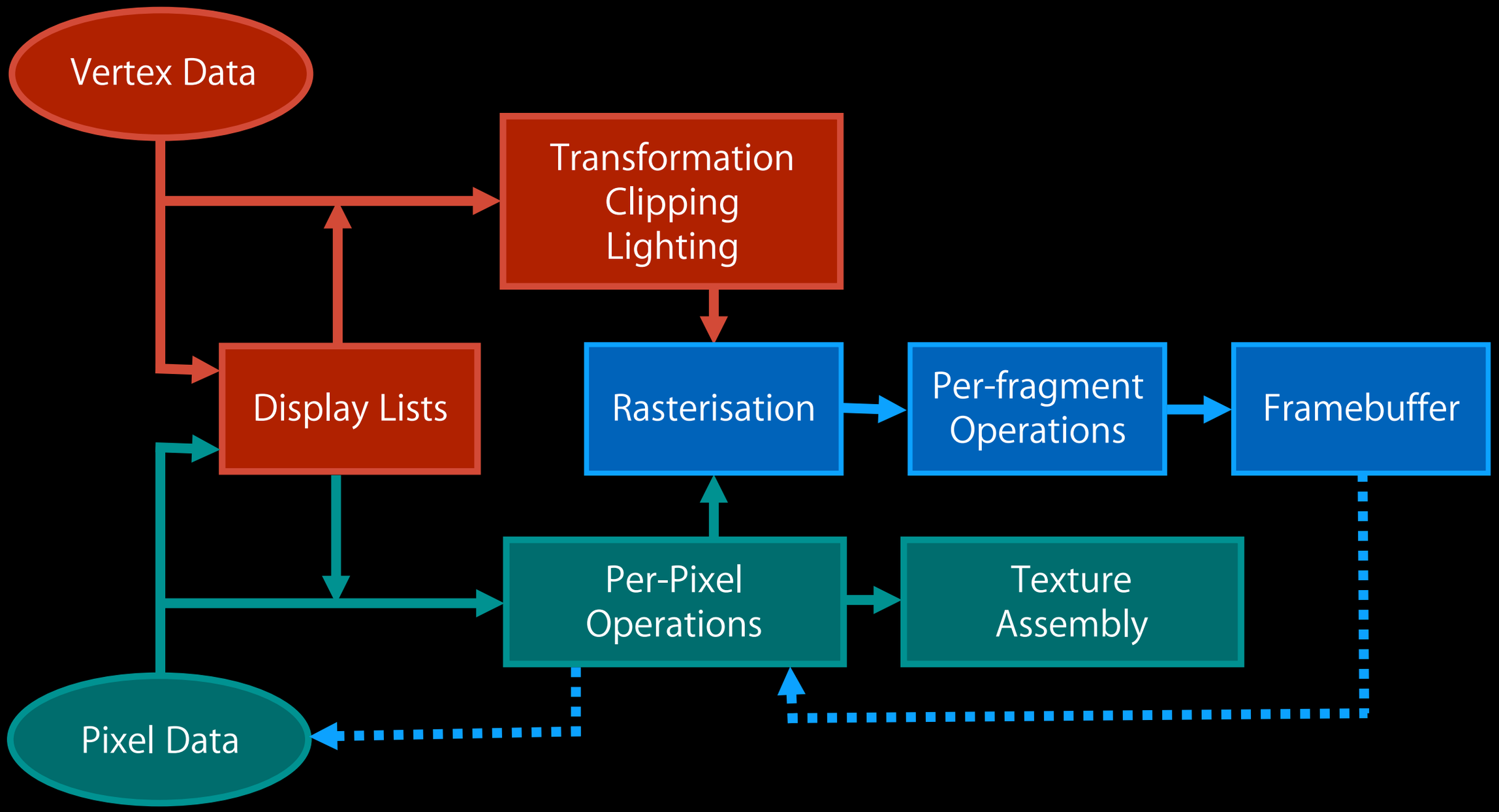
- Avoid calling glFlush
- Never call glFinish
- Avoid glReadPixels
- Avoid immediate mode drawing (glBegin/glEnd)
  - Except in display lists
- Minimize state changes
  - Group drawing according to state

Optimizing Texture Uploads

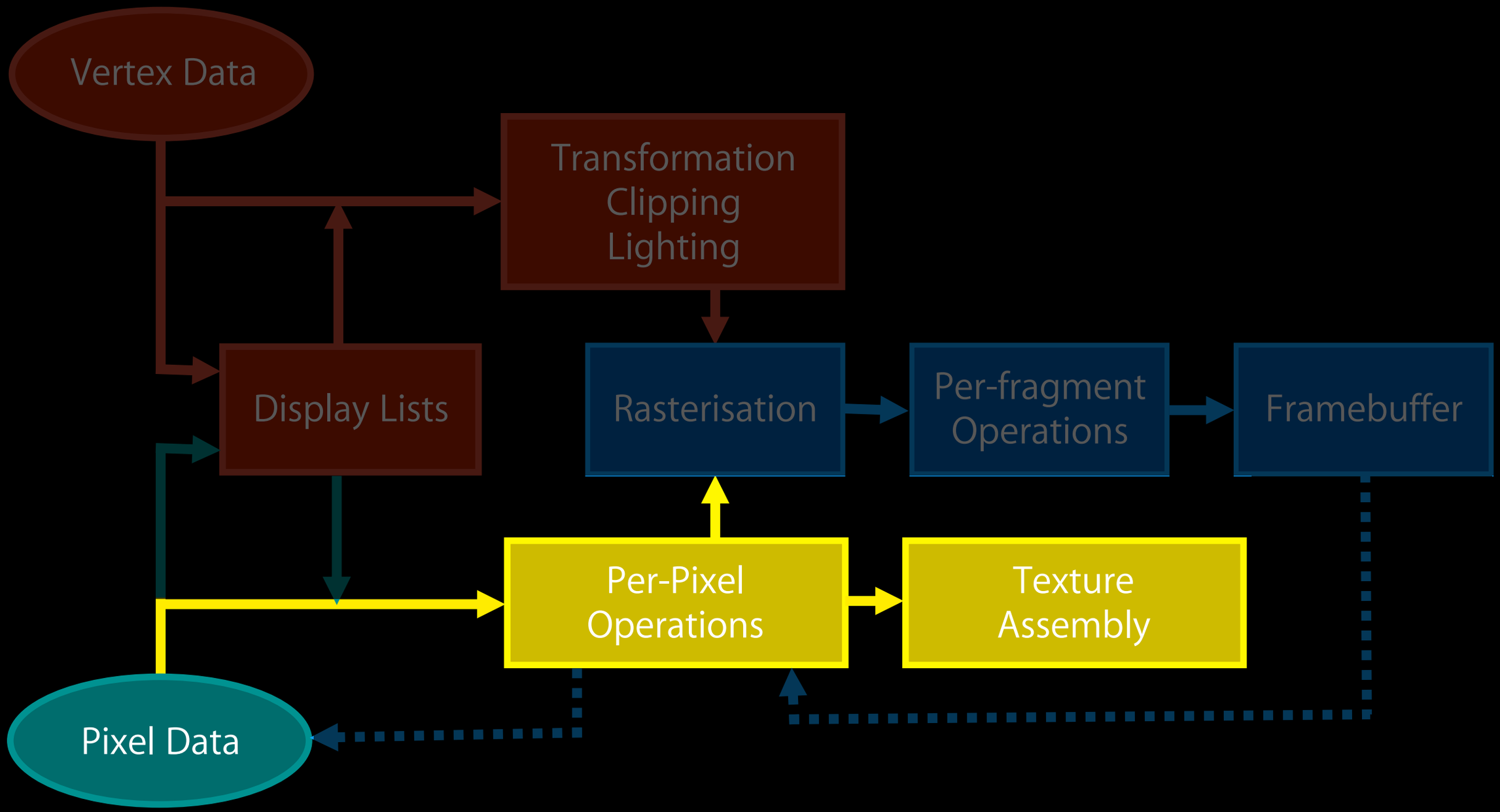
- Texture pipeline overview
- Texture optimization basics
- OpenGL texture extensions
  - Power-of-two textures
  - Non power-of-two textures

Sample Code: Texture Range

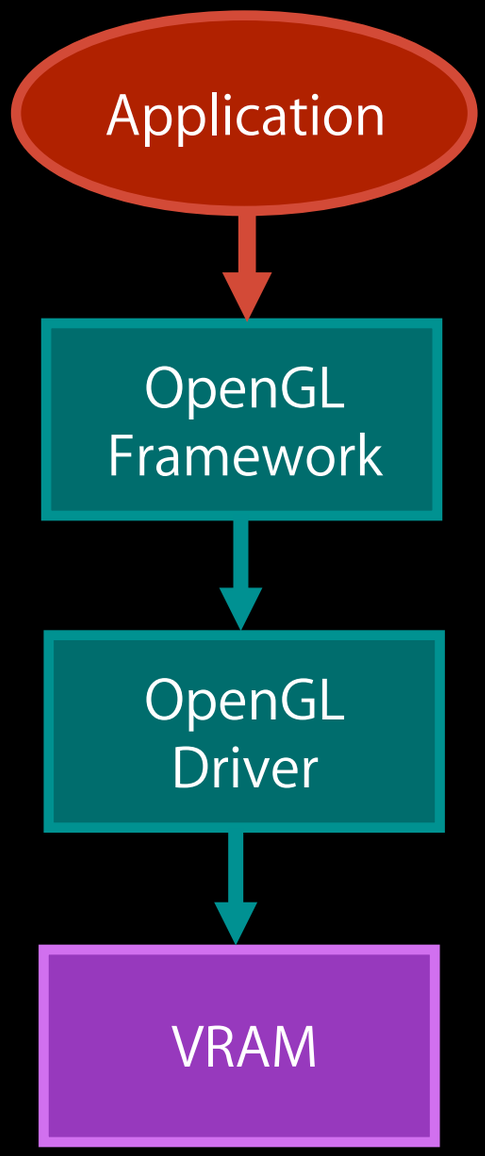
OpenGL Pipeline



Pixel Pipeline



Texture Upload Overview



Texture Optimization Basics

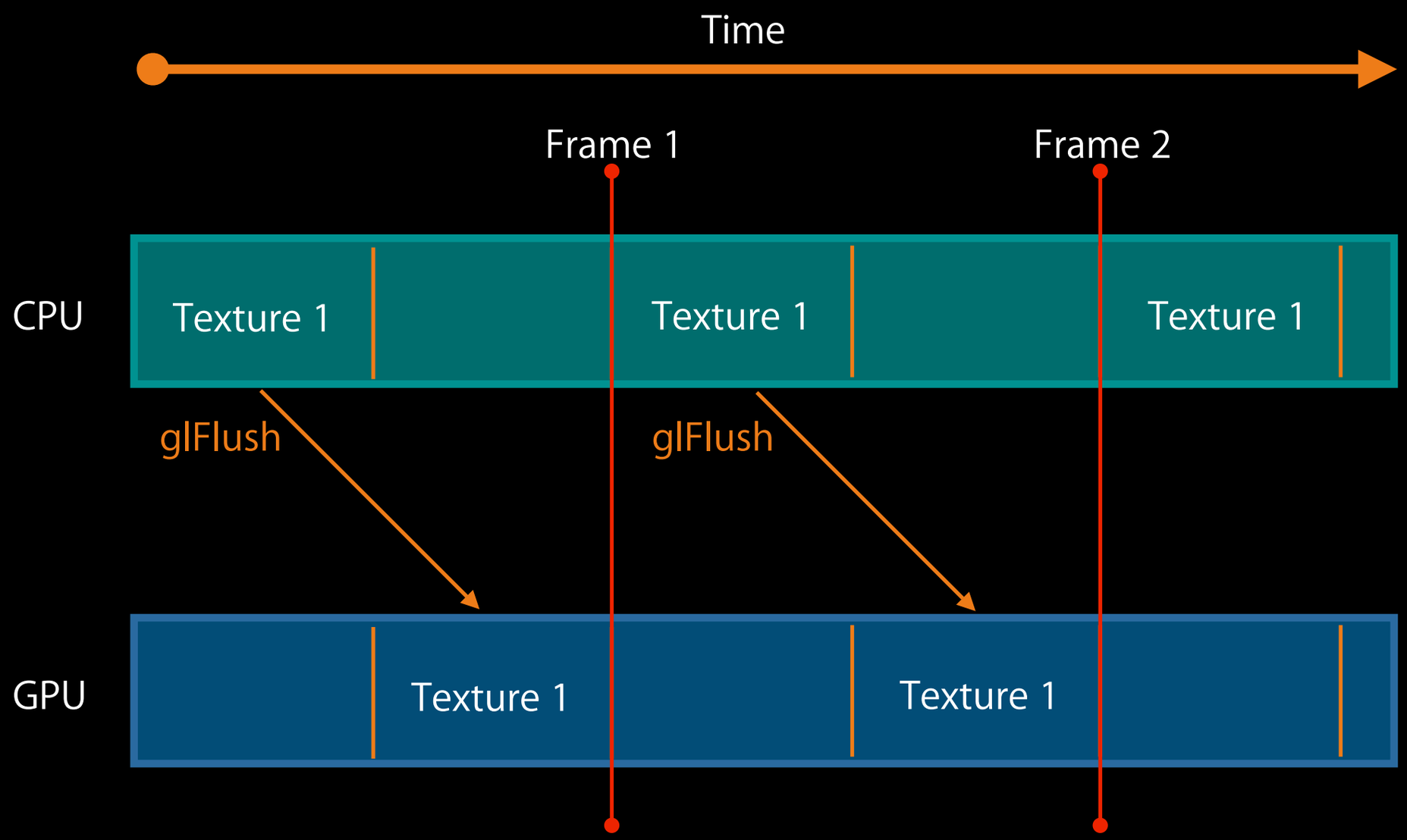
- Minimize CPU copies and conversions
  - Correct setup will give you zero CPU copies
- Optimal Texture types and formats

```
GL_BGRA, GL_UNSIGNED_INT_8_8_8_8_REV
GL_BGRA, GL_UNSIGNED_SHORT_1_5_5_5_REV
YCBCR_422_APPLE, UNSIGNED_SHORT_8_8_REV_APPLE
```

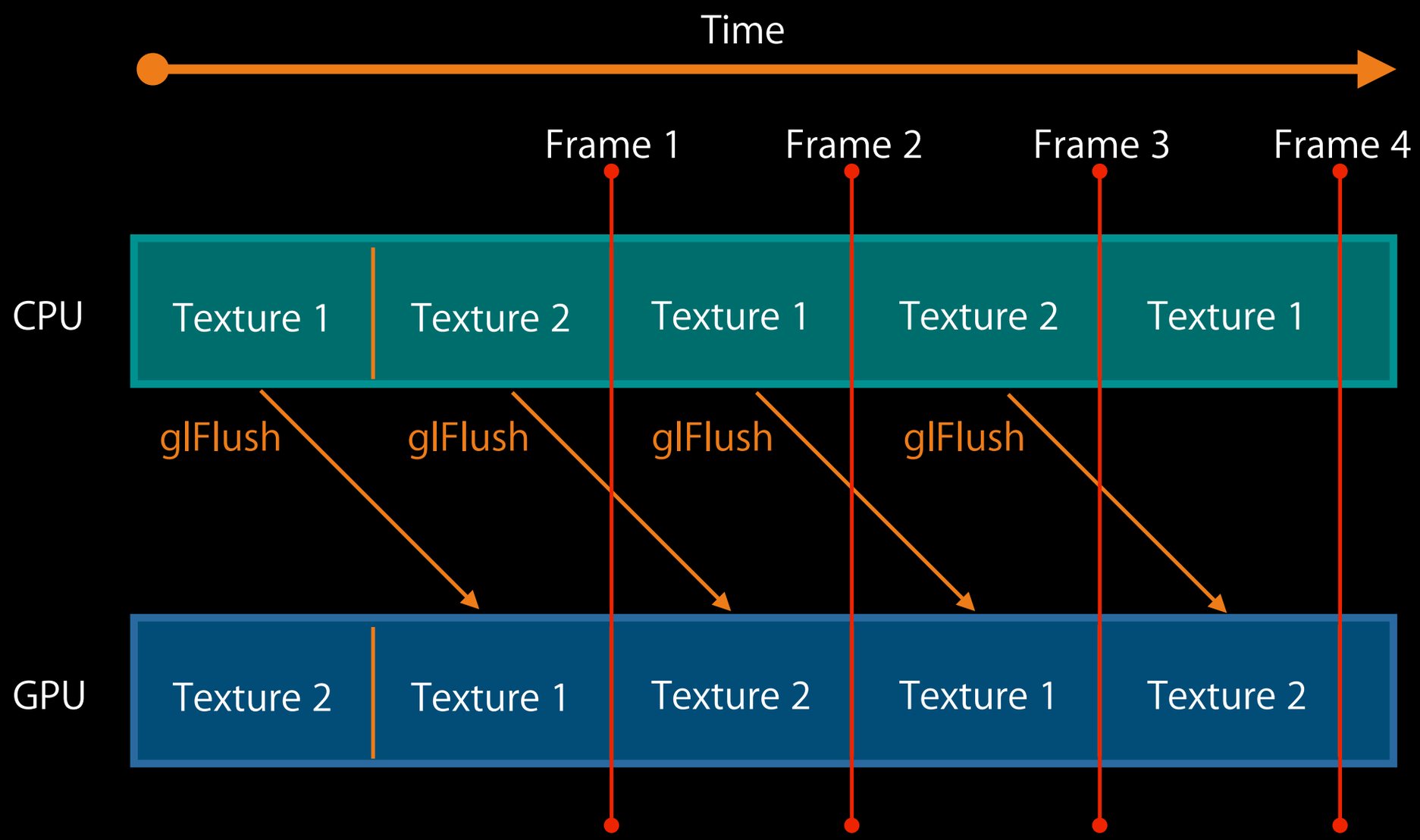
Texture Optimization Basics

- Double buffer texture data
  - glFlush between draws

Non-Double Buffered Texture Data



Double Buffered Texture Data



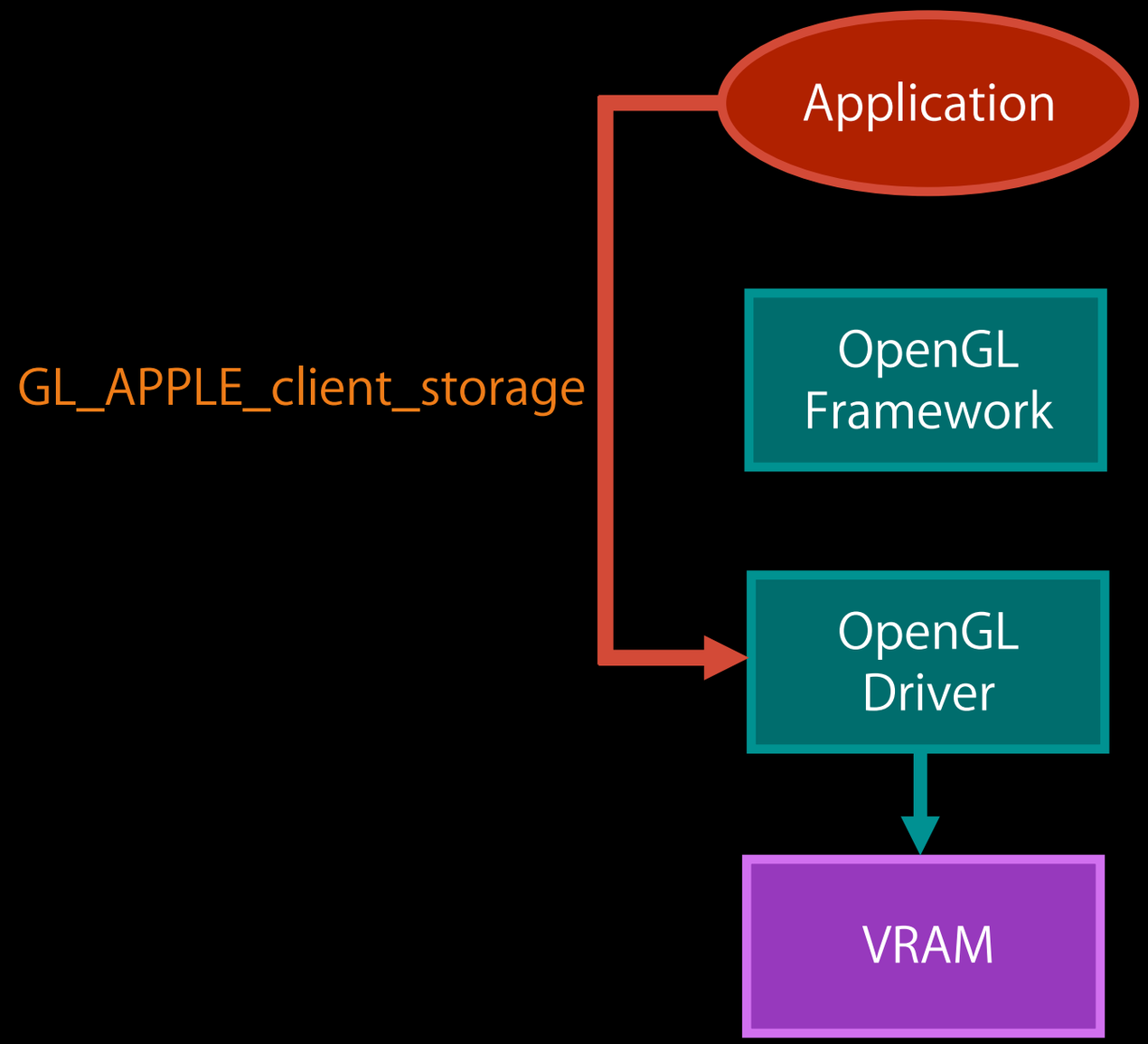
OpenGL Texture Extensions

- GL_APPLE_client_storage
  - Requires application to retain texture storage
- GL_APPLE_texture_range

  GL_STORAGE_CACHED_APPLE

  GL_STORAGE_SHARED_APPLE
- GL_EXT_texture_rectangle
  - Required for some hardware for direct DMA

Client Storage

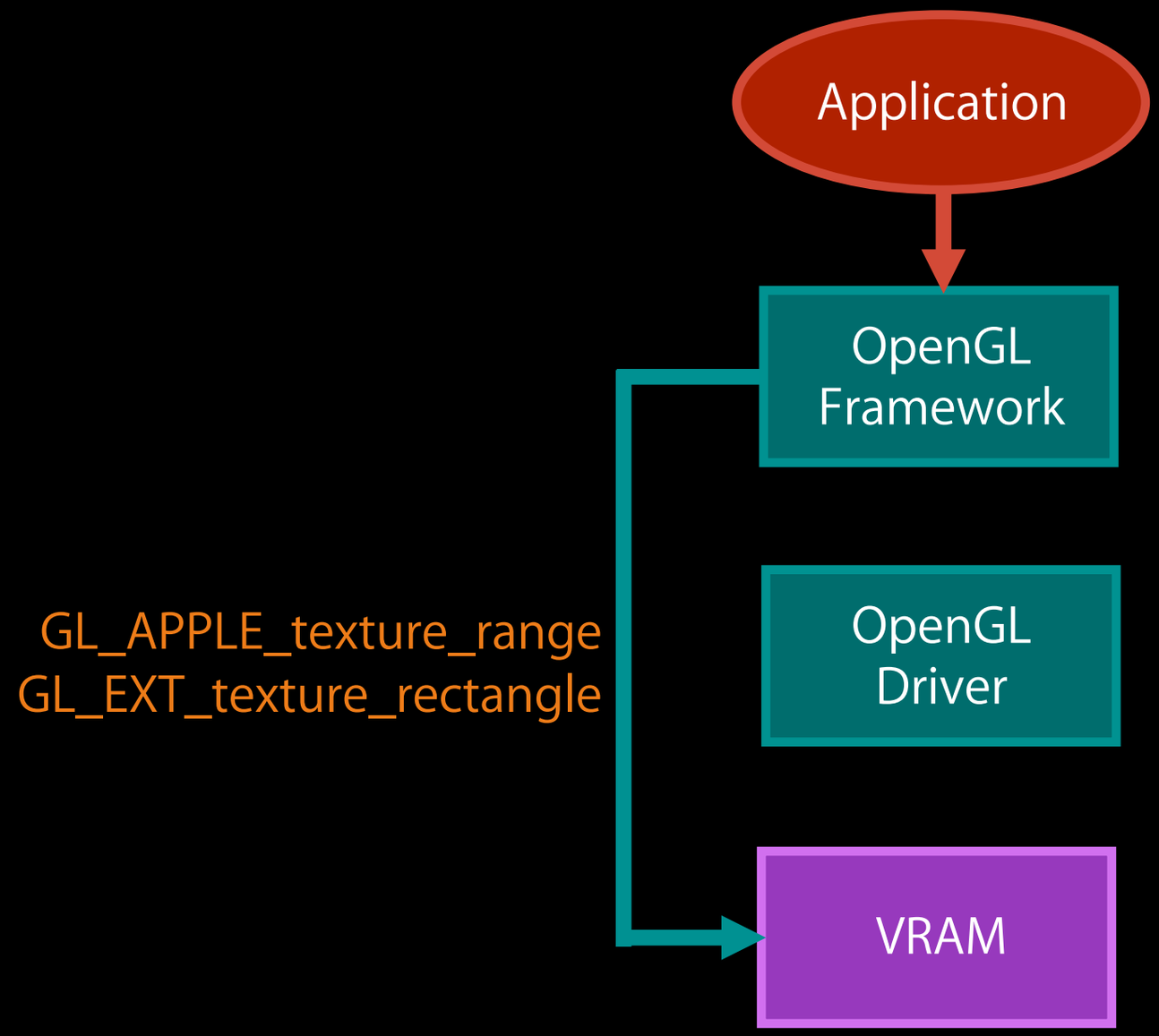


Client Storage

- Usable for power-of-two and non power-of-two

```
glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_TRUE);
```

Texture Range and Rectangle Texture



Texture Range and Rectangle Texture

- Texture range cached hint
- Usable for non power-of-two

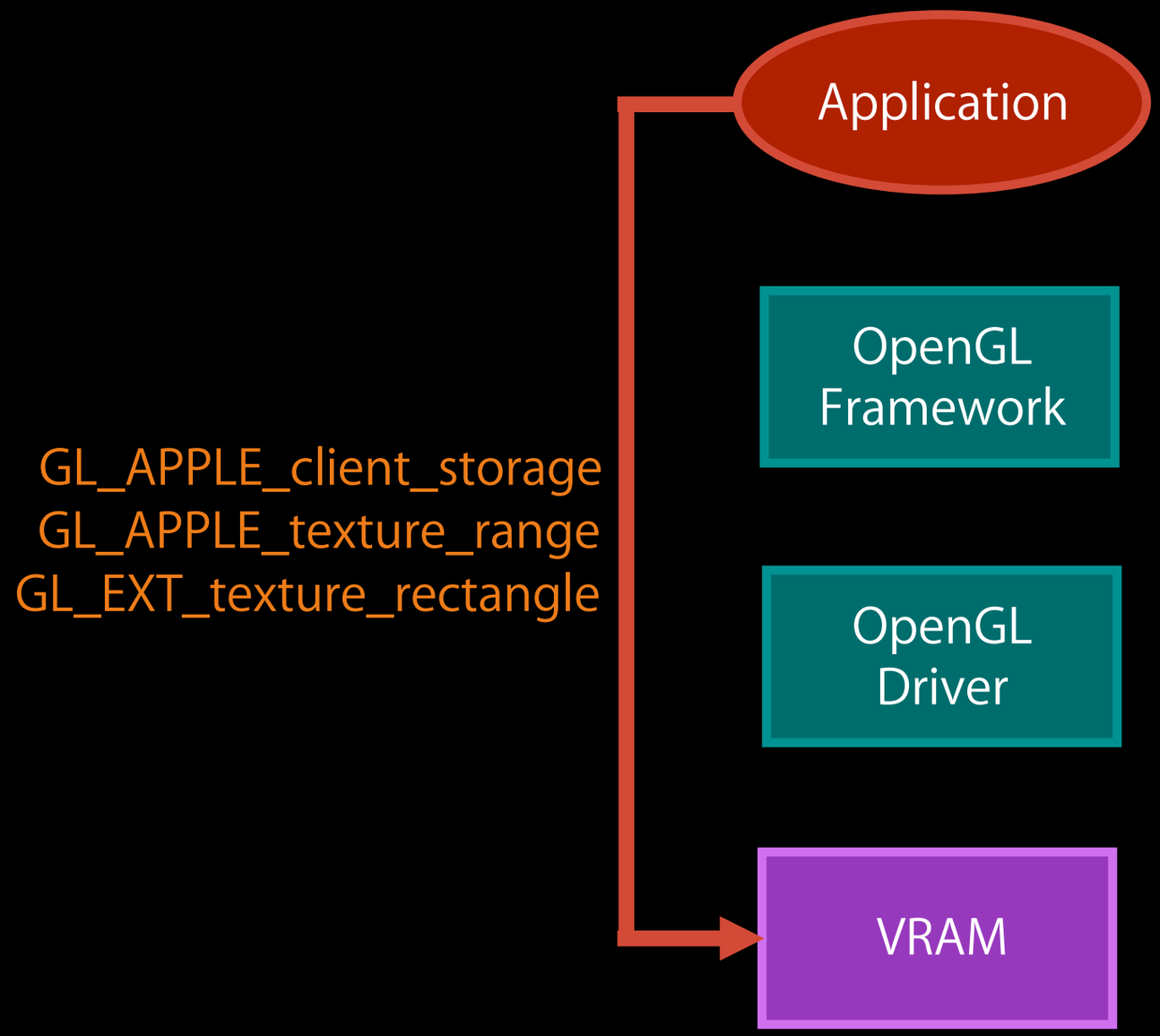
```
glTexParameteri(GL_TEXTURE_RECTANGLE_EXT,
   GL_TEXTURE_STORAGE_HINT_APPLE,
   GL_STORAGE_CACHED_APPLE);
```

Texture Range and Rectangle Texture

- Texture range shared hint
- Usable for non power-of-two

```
glTexParameteri(GL_TEXTURE_RECTANGLE_EXT,
   GL_TEXTURE_STORAGE_HINT_APPLE,
   GL_STORAGE_SHARED_APPLE);
```

OpenGL Texture Extensions



OpenGL Texture Extensions

- Texture setup for non power-of-two

```
glBindTexture(GL_TEXTURE_RECTANGLE_EXT, id);

glTexParameteri(GL_TEXTURE_RECTANGLE_EXT,
   GL_TEXTURE_STORAGE_HINT_APPLE,
   GL_STORAGE_CACHED_APPLE);

glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_TRUE);

glTexImage2D(GL_TEXTURE_RECTANGLE_EXT,
   0, GL_RGBA, sizex, sizey, GL_BGRA,
   GL_UNSIGNED_INT_8_8_8_8_REV,
   image_ptr);
```

Sample Code: Texture Range

OpenGL Texture Extensions

- Texture setup for power-of-two

```
glBindTexture(GL_TEXTURE_2D, id);

glTexParameteri(GL_TEXTURE_2D,
   GL_TEXTURE_STORAGE_HINT_APPLE,
   GL_STORAGE_CACHED_APPLE);

glPixelStorei(GL_UNPACK_CLIENT_STORAGE_APPLE, GL_TRUE);

glTexImage2D(GL_TEXTURE_2D,
   0, GL_RGBA, sizex, sizey, GL_BGRA,
   GL_UNSIGNED_INT_8_8_8_8_REV,
   image_ptr);
```

Sample Code: Texture Range

Demo

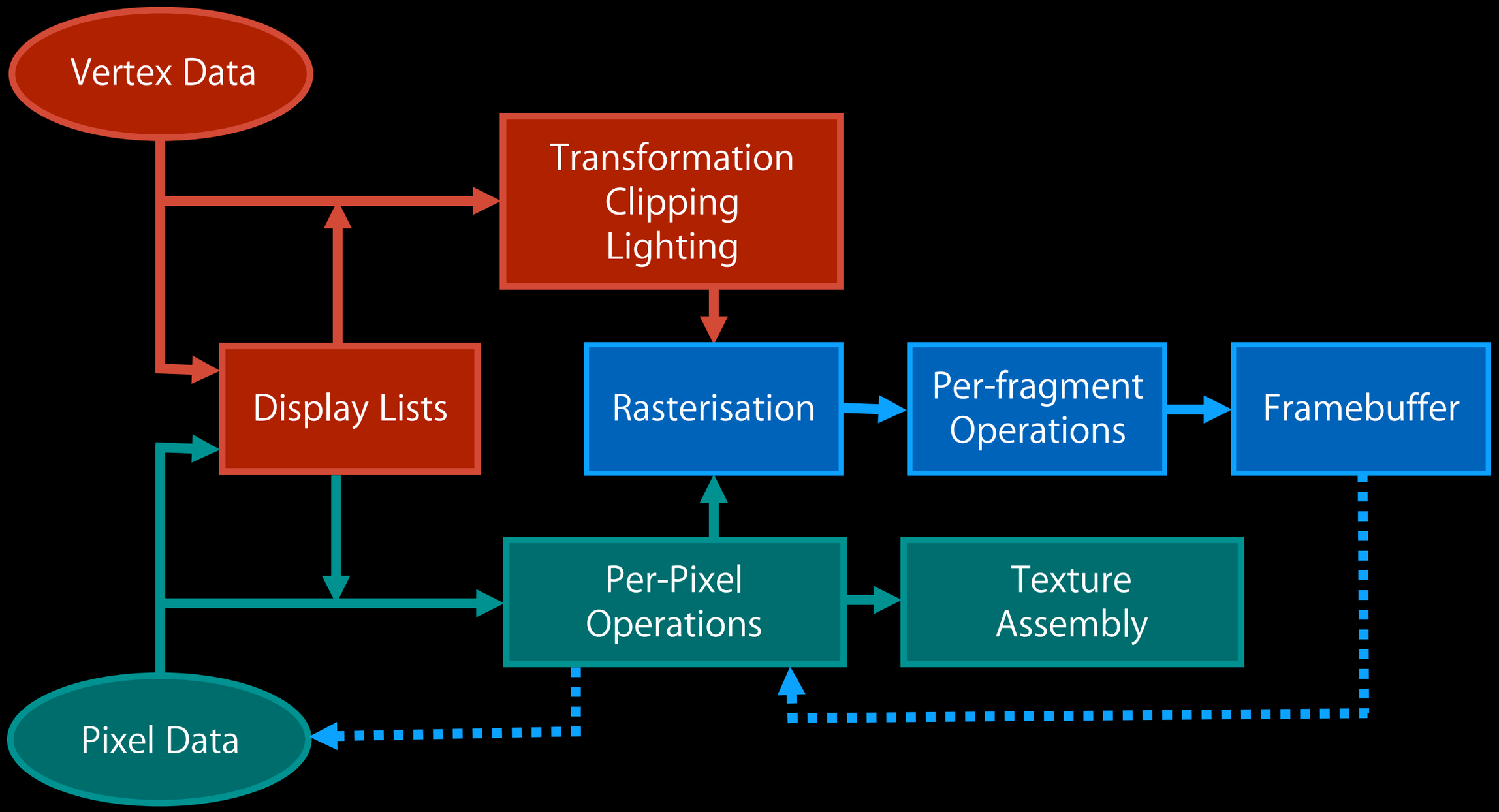
Texture Range Sample Code

Optimizing Vertex Throughput

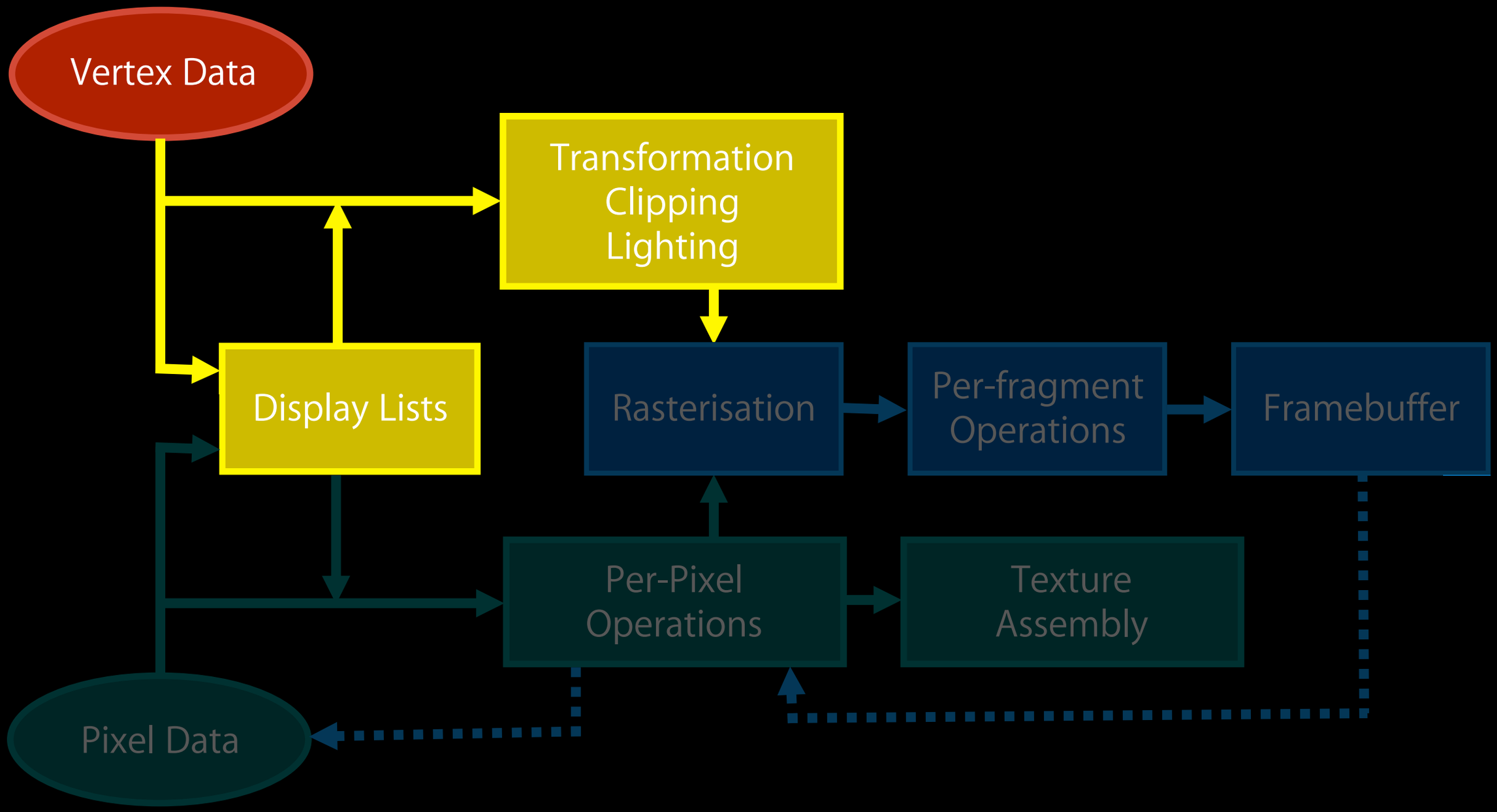
- Vertex pipeline overview
- Vertex optimization basics
- OpenGL vertex APIs
  - Dynamic vertex data
  - Static vertex data
  - Display lists

Sample Code: Vertex Optimization

OpenGL Pipeline



Vertex Pipeline



Vertex Optimization Basics

- Minimize CPU copies
  - Use GLfloat for vertex data type
- Use vertex arrays

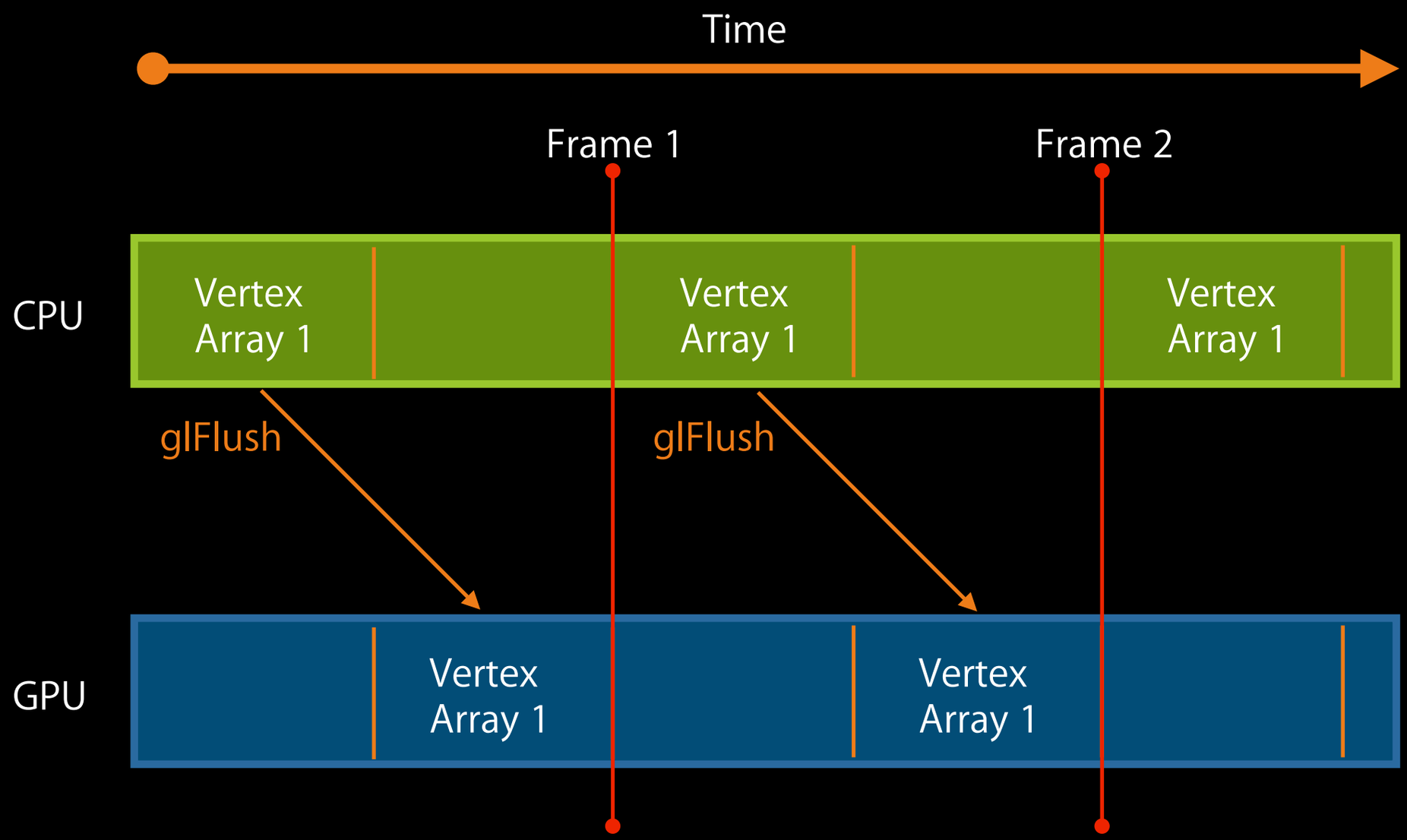
```
glDrawRangeElements
glDrawElements
glDrawArrays
```

- Maximize vertices per draw command
- Cache static data in VRAM
- Use vertex programs to offload CPU work

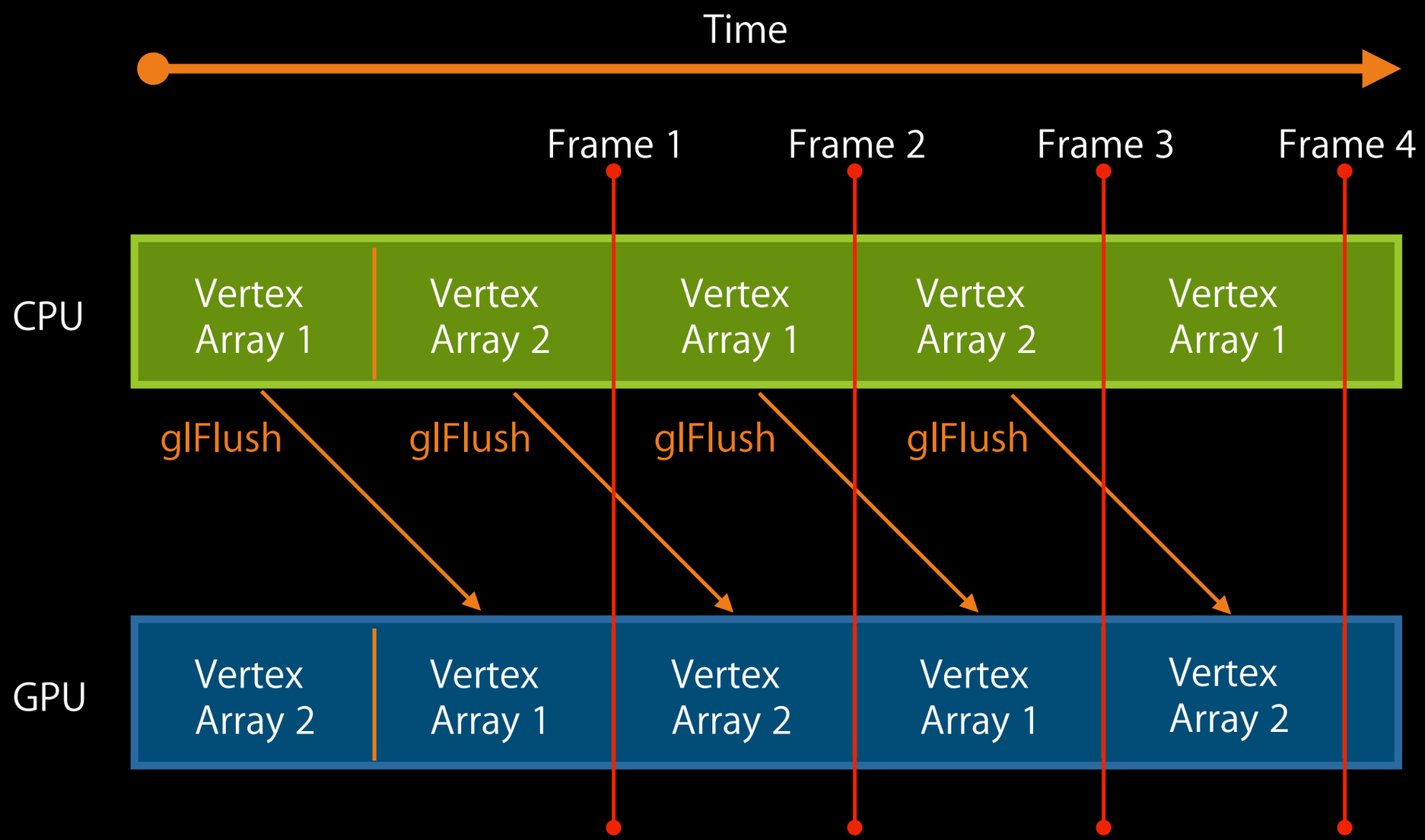
Vertex Optimization Basics

- Double buffer vertex data
  - glFlush between draws

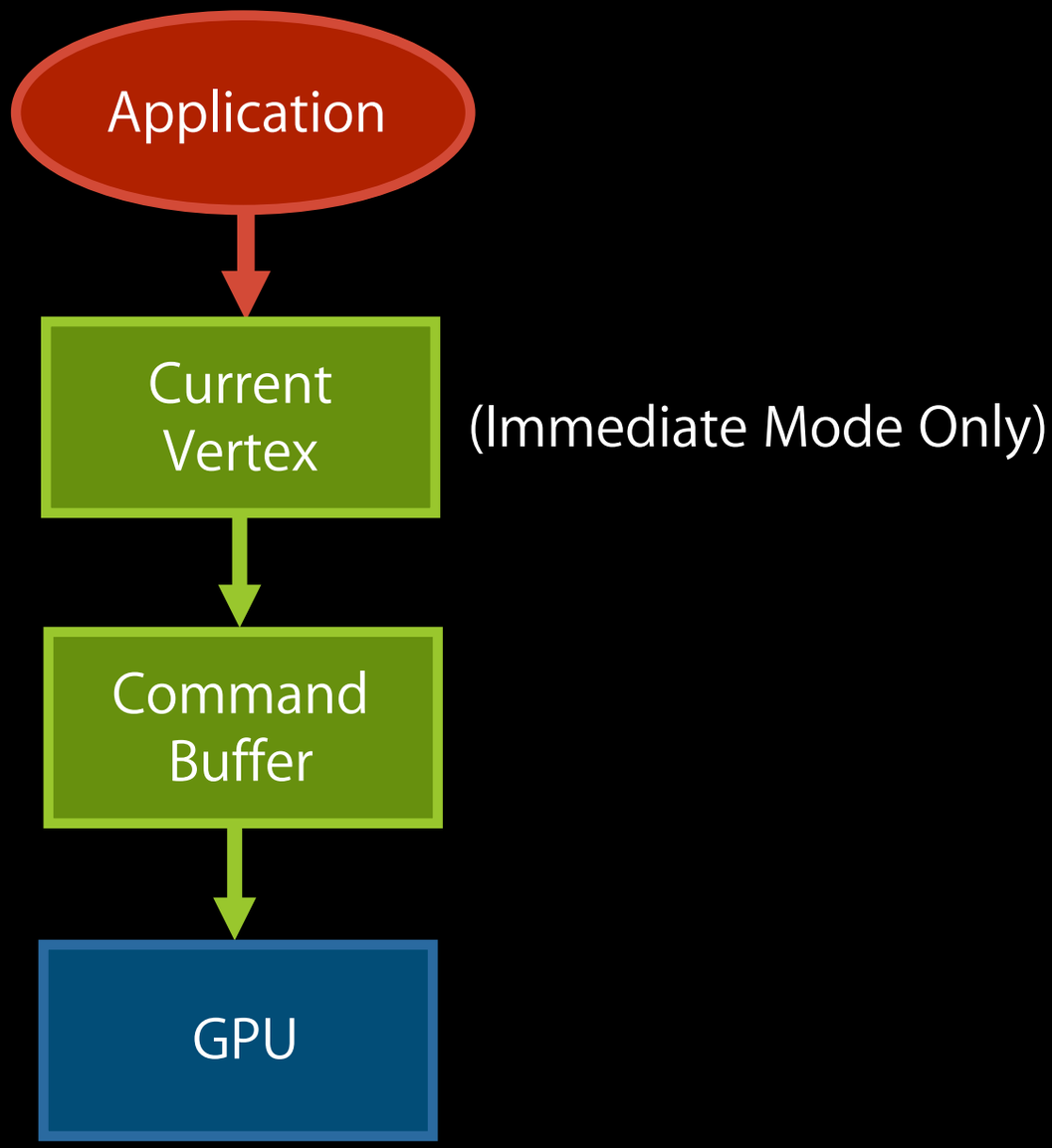
Non-Double Buffered Vertex Data



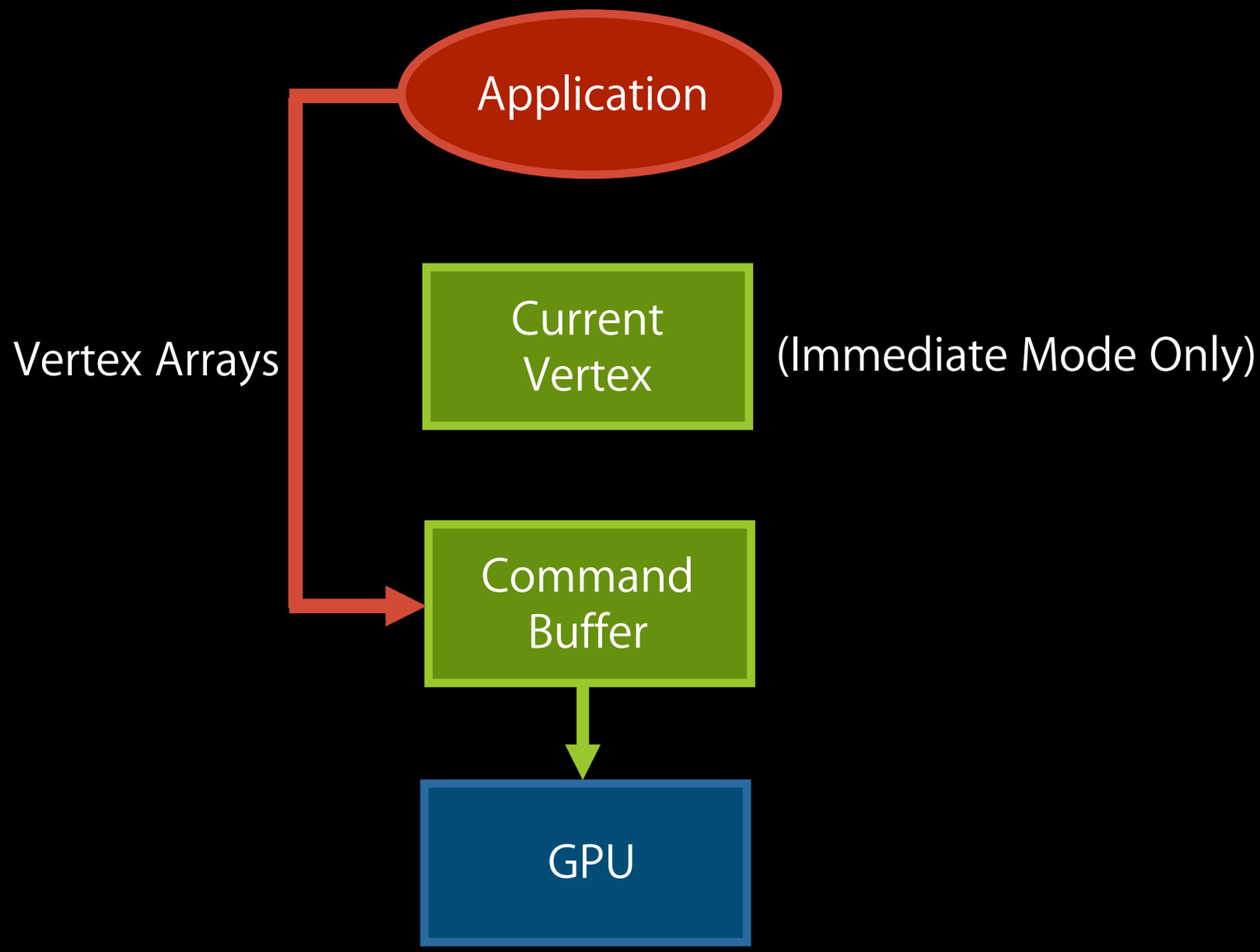
Double Buffered Vertex Data



Vertex Pipeline Overview



Vertex Arrays



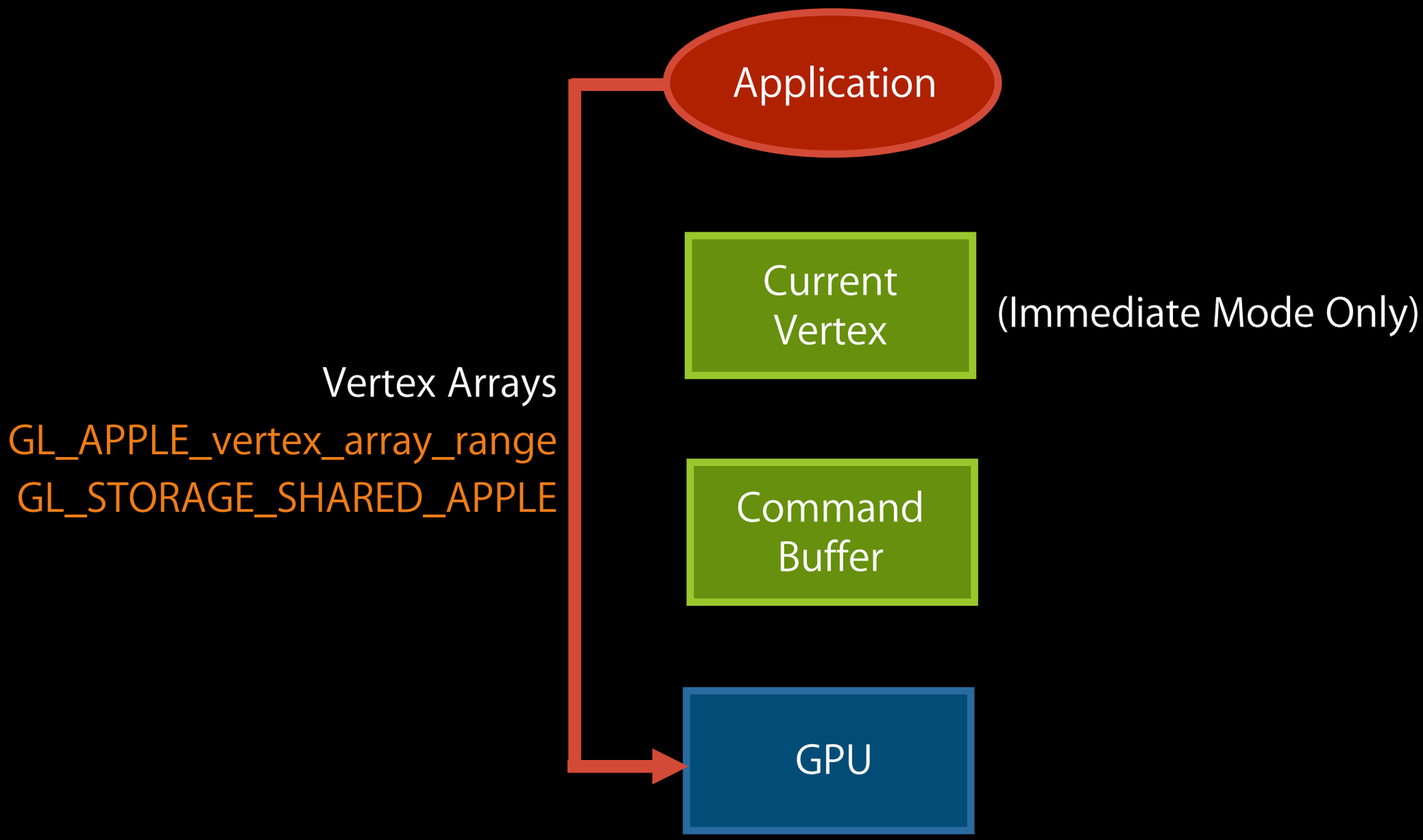
Dynamic Vertex Data

- GL_APPLE_vertex_array_range

  `GL_STORAGE_SHARED_APPLE`

Vertex Array Range

- Dynamic data setup



Vertex Array Range

- Dynamic data setup

```
glEnableClientState(GL_VERTEX_ARRAY);

glVertexPointer(3, GL_FLOAT, stride, vertex_ptr);

glVertexArrayRangeApple(array_size, vertex_ptr);

glFlushVertexArrayRangeAPPLE(array_size, vertex_ptr);
```

Static Vertex Data

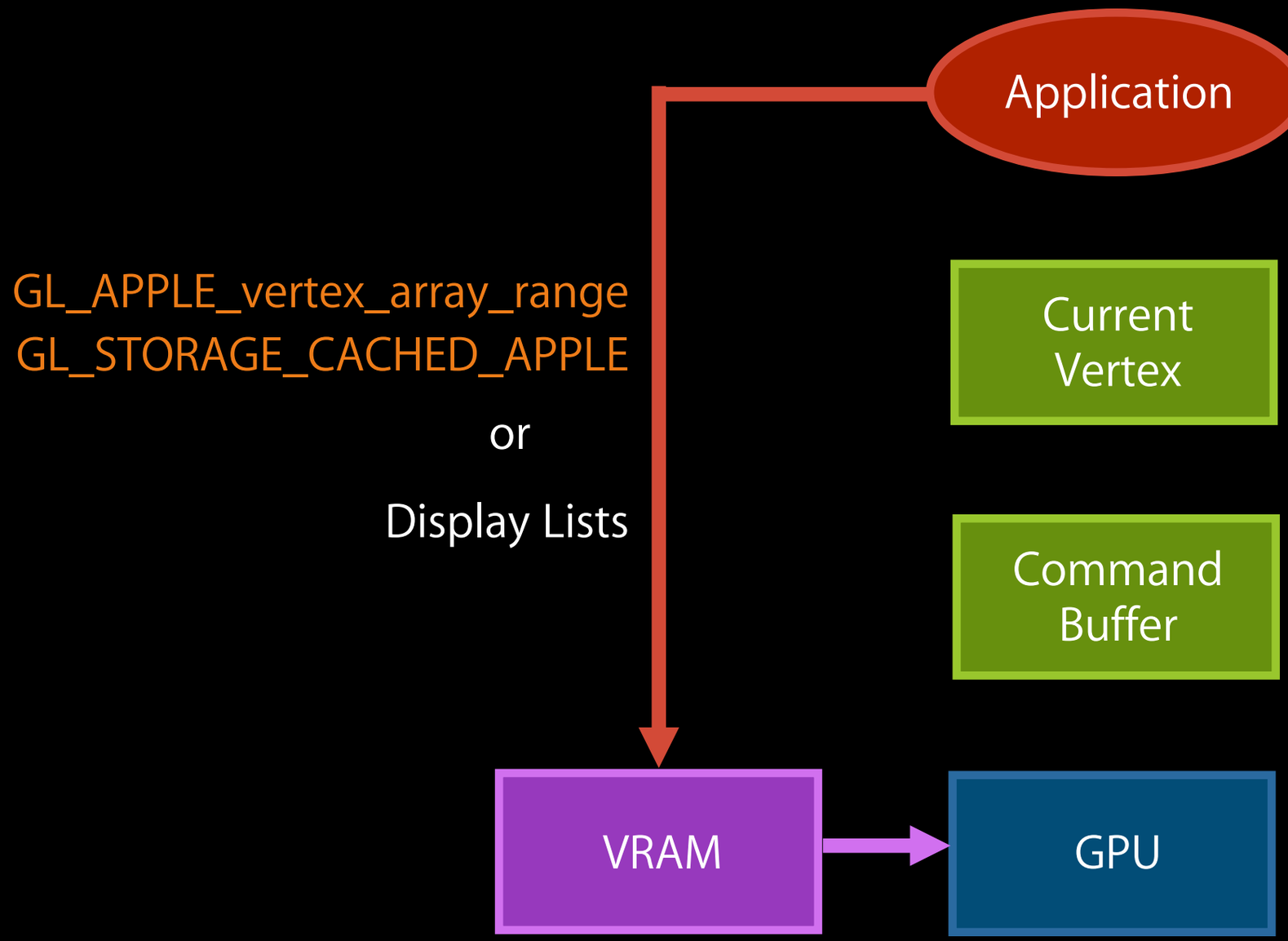
- GL_APPLE_vertex_array_range

  `GL_STORAGE_CACHED_APPLE`

- Display Lists
  - glBegin/glEnd
  - Use consistent vertex data
  - Must have more than 16 vertices in the list

Vertex Array Range/Display Lists

- Static data setup



Vertex Array Range

- Static data setup

```
glEnableClientState(GL_VERTEX_ARRAY);

glVertexPointer(3, GL_FLOAT, stride, vertex_ptr);

glVertexArrayParameteriAPPLE(
   GL_VERTEX_ARRAY_STORAGE_HINT_APPLE,
   GL_STORAGE_CACHED_APPLE);

glVertexArrayRangeApple(array_size, vertex_ptr);

glFlushVertexArrayRangeAPPLE(array_size, vertex_ptr);
```

Sample Code: Vertex Optimization

Display Lists

- Static data using display lists

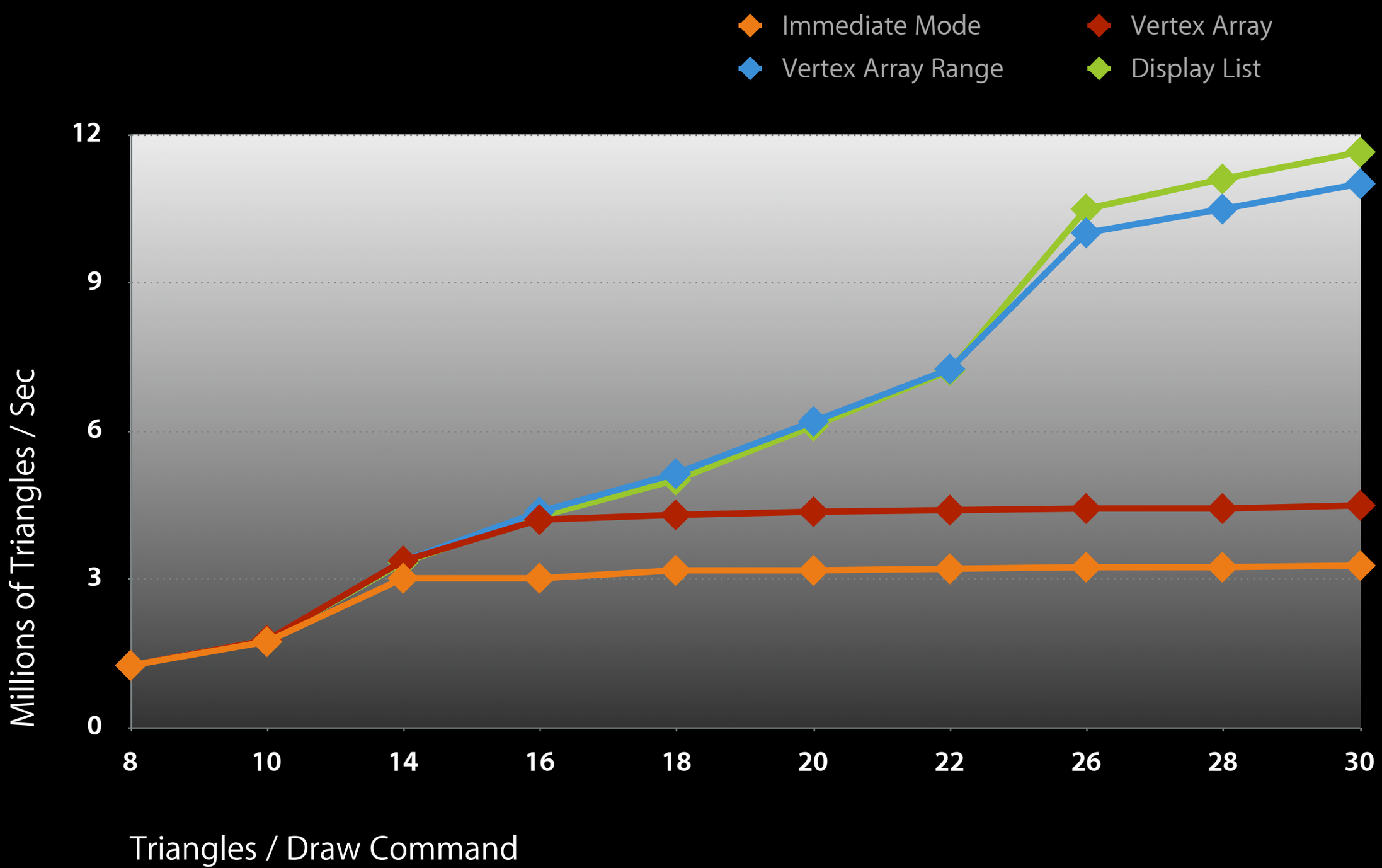
```
glBeginList(list_name);

   (draw here)

glEndList();
```

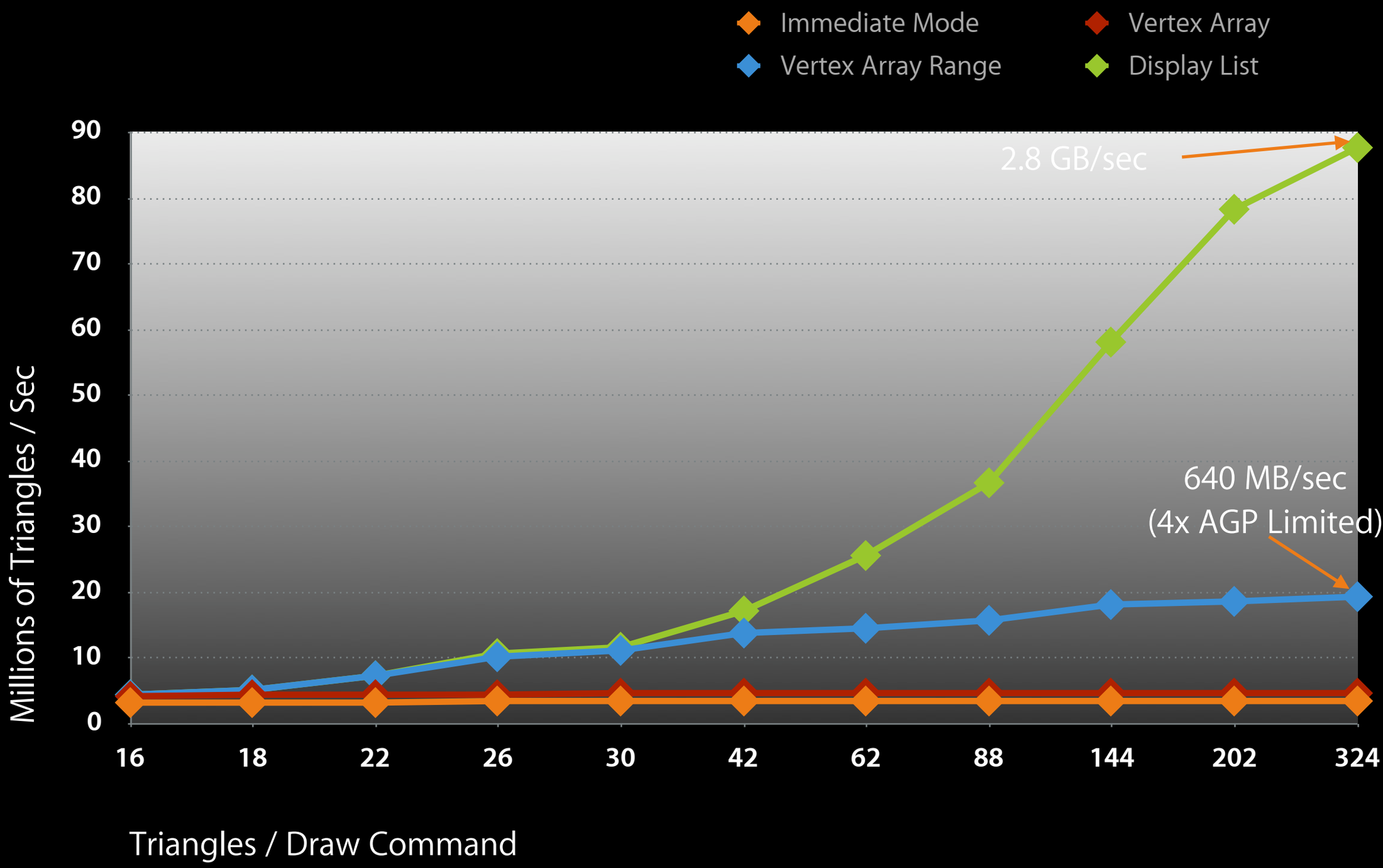
Sample Code: Vertex Optimization

Low Vertex Count Performance



Results from a Dual 1.42 GHz G4 with an R300 (results will vary)

High Vertex Count Performance



Results from a Dual 1.42 GHz G4 with an R300 (results will vary)

Demo

Vertex Optimization Sample Code

Optimizing for One Shot Images

- Use glDrawPixels
  - Best with small images
- Recommend textures for larger images
  - Greater than 128x128

Sample Code: Draw Pixels

Optimizing for One Shot Images

- Disable complex state for best performance

```
glDisable(GL_DITHER);
glDisable(GL_ALPHA_TEST);
glDisable(GL_BLEND);
glDisable(GL_STENCIL_TEST);
glDisable(GL_FOG);
glDisable(GL_TEXTURE_2D);
glDisable(GL_DEPTH_TEST);
glPixelZoom(1.0, 1.0);
```

Optimizing for One Shot Images

- glDrawPixels sample

```
void initWithFrame:(NSRect)frameRect
{
   glDisable(GL_DITHER);
   /* Disable other state if needed */
}

void DrawRect:(NSRect) aRect
{
   glDrawPixels(width, height, GL_BGRA,
      GL_UNSIGNED_INT_8_8_8_8_REV, image);
}
```

Demo

Draw Pixels Sample Code

Optimizing Pixel Copy Operations

- Use glCopyPixels
  - Correct setup will give you a VRAM to VRAM copy
- Use AUX buffers for temporary pixel storage

  `NSOpenGLPFAAuxBuffers`

  `NSOpenGLPFAAuxDepthStencil`

Optimizing Pixel Copies

- Disable complex state for VRAM to VRAM copy

```
glDisable(GL_DITHER);
glDisable(GL_ALPHA_TEST);
glDisable(GL_BLEND);
glDisable(GL_STENCIL_TEST);
glDisable(GL_FOG);
glDisable(GL_TEXTURE_2D);
glDisable(GL_DEPTH_TEST);
glPixelZoom(1.0, 1.0);
```

Optimizing for One Shot Images

- glCopyPixels sample

```
void initWithFrame:(NSRect)frameRect
{
   glDisable(GL_DITHER);
   /* Disable other state if needed */
}

void DrawRect:(NSRect) aRect
{
   glDrawBuffer(GL_BACK);
   glReadBuffer(GL_AUX0);

   glCopyPixels(x, y, width, height, GL_COLOR);
}
```

Sample Code: Draw Pixels

Using Threads With OpenGL

- Rules for threading OpenGL
- Divisions of work
- Sharing data between contexts
- Thread synchronization

Sample Code: Vertex Optimization

Rules for Threading OpenGL

- Can’t do
  - Per context re-entrance

- Can do
  - Context sharing across threads
  - Surface sharing across contexts

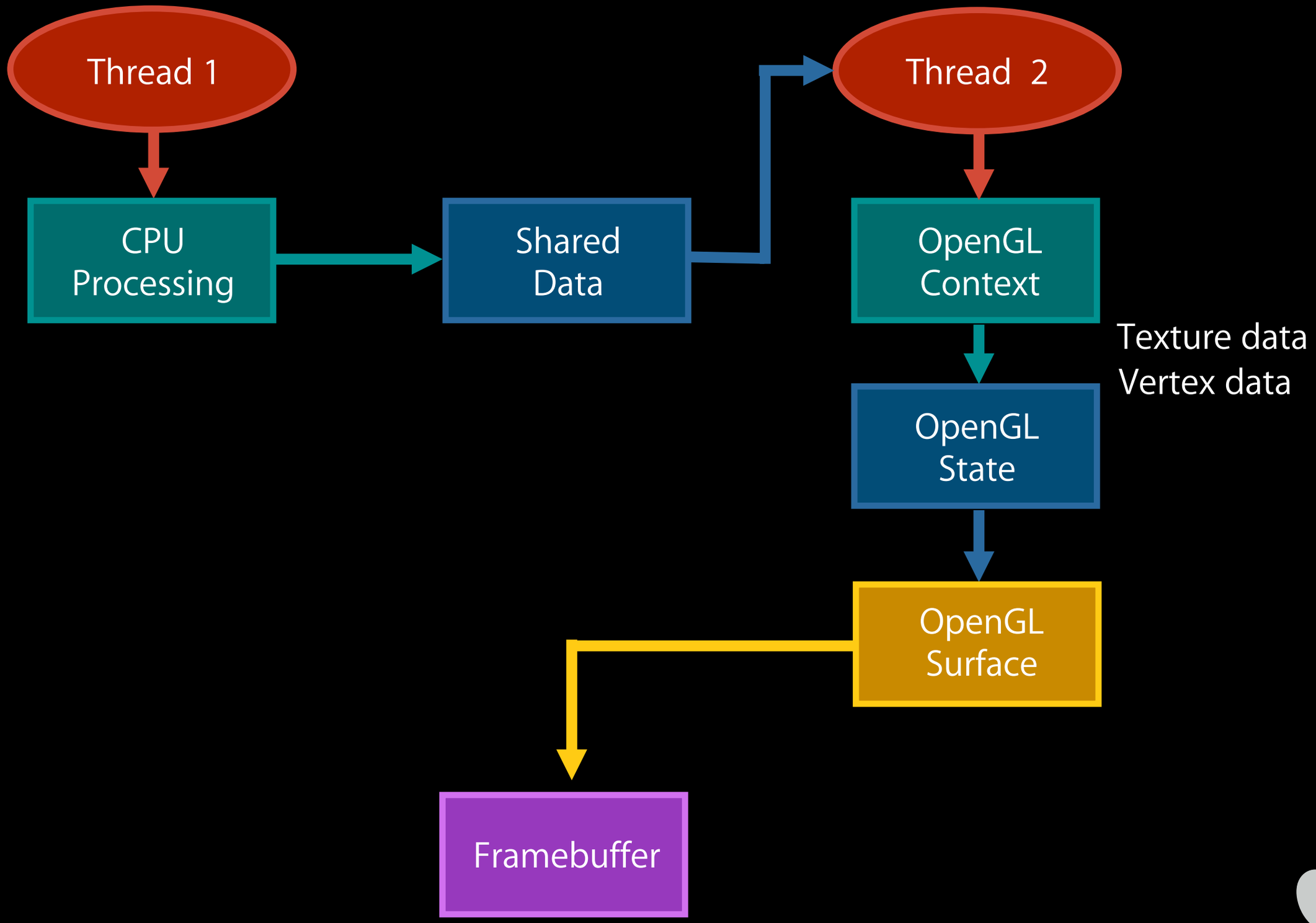
Divisions of Work

- Move OpenGL onto a separate thread
- Split OpenGL texture and vertex processing
- Split OpenGL surface processing

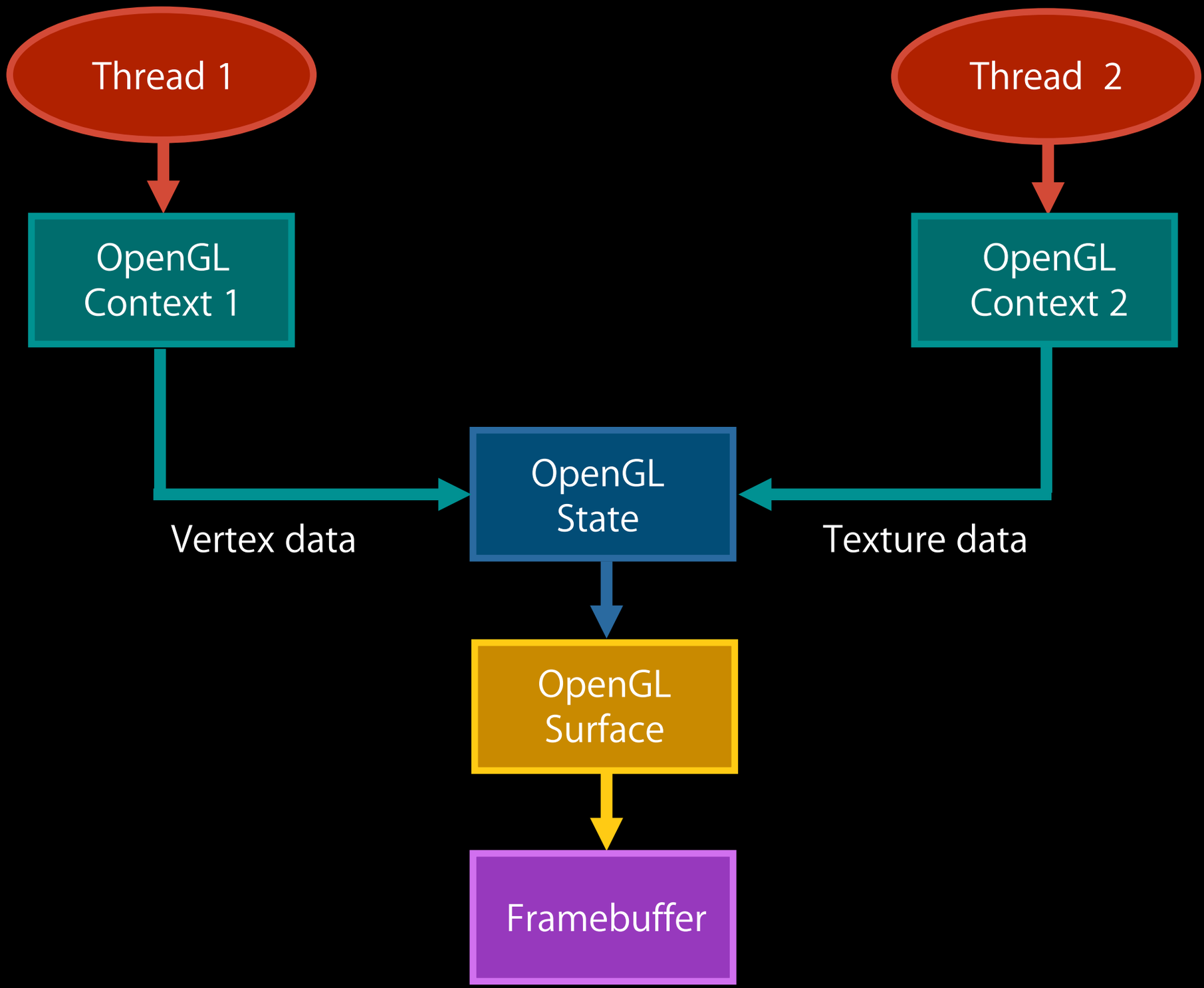
Sharing Data Between Contexts

- Share OpenGL context state
  - Display lists
  - Textures
  - Vertex and Fragment Programs
  - Vertex Array Objects
- Share an OpenGL surface
  - VRAM buffer

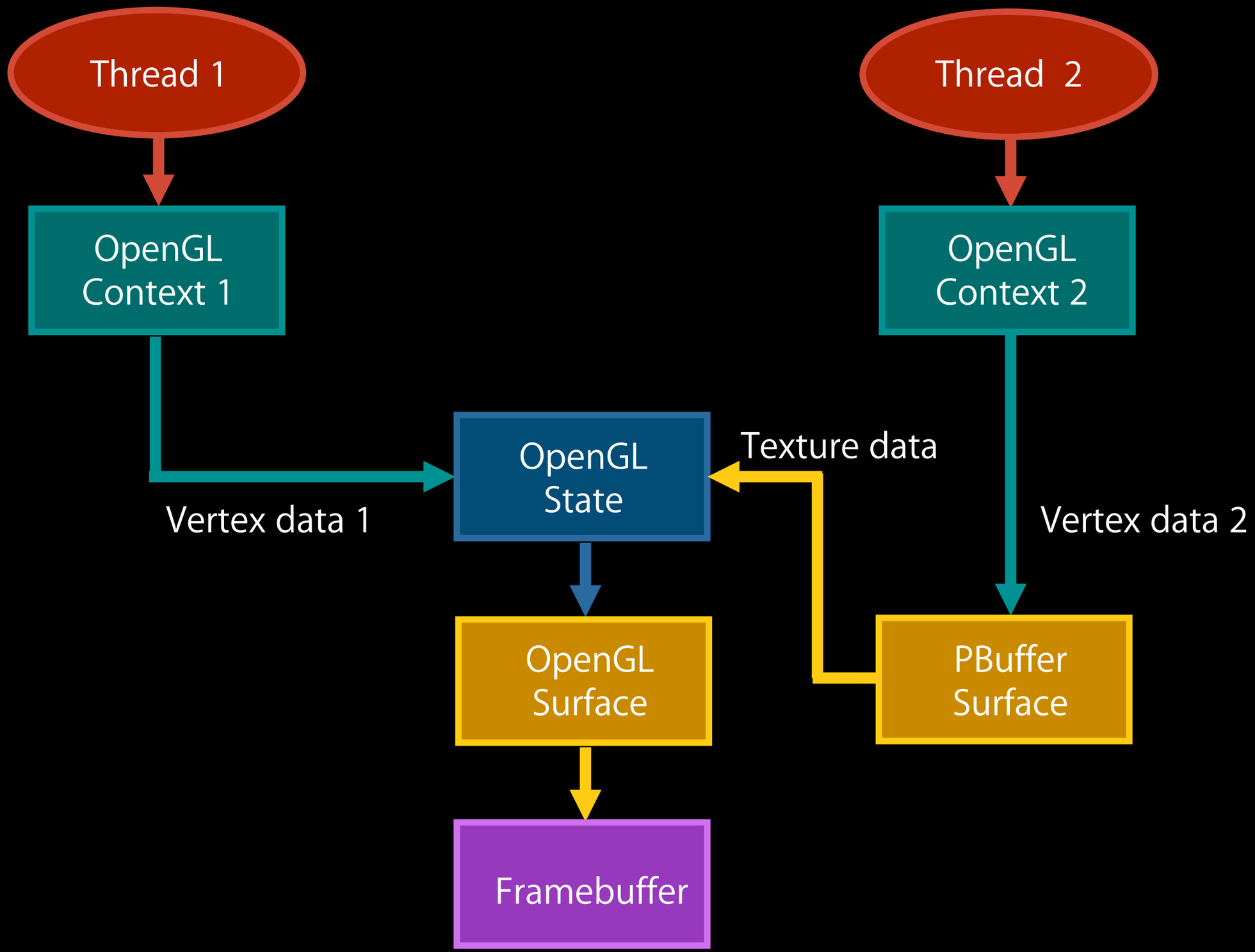
Move OpenGL Onto a Separate Thread



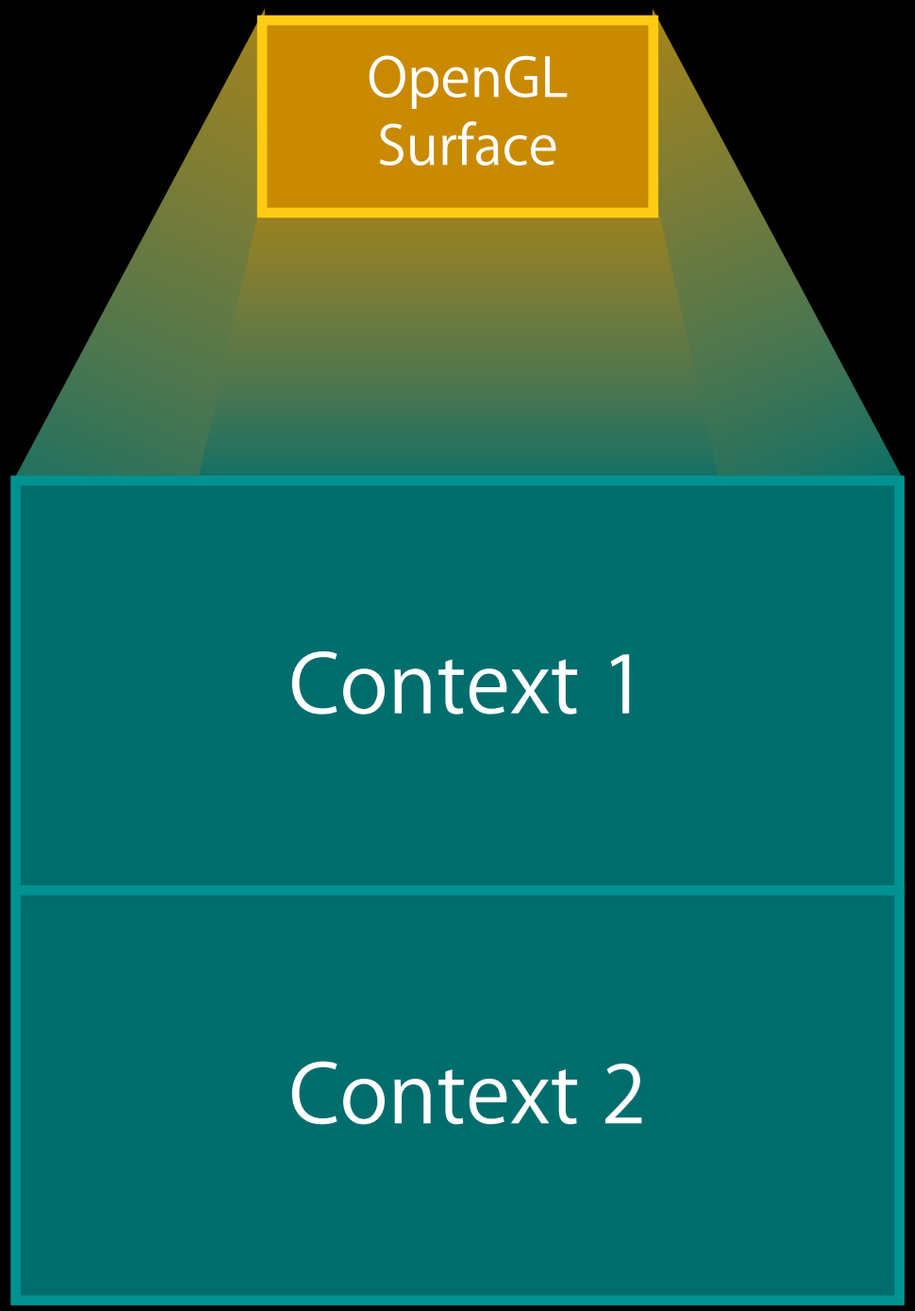
Split Texture and Vertex Processing



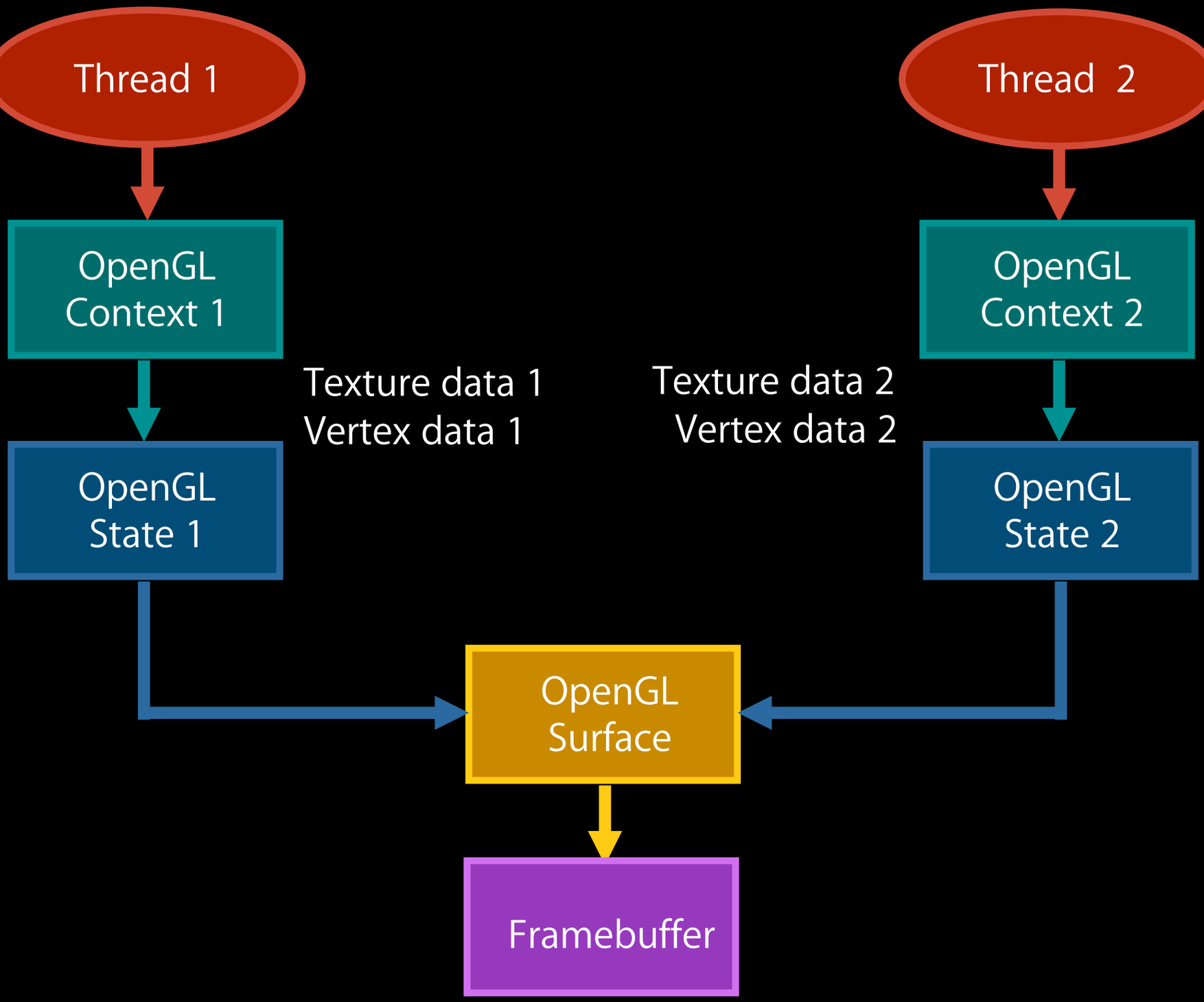
Split Texture and Vertex Processing



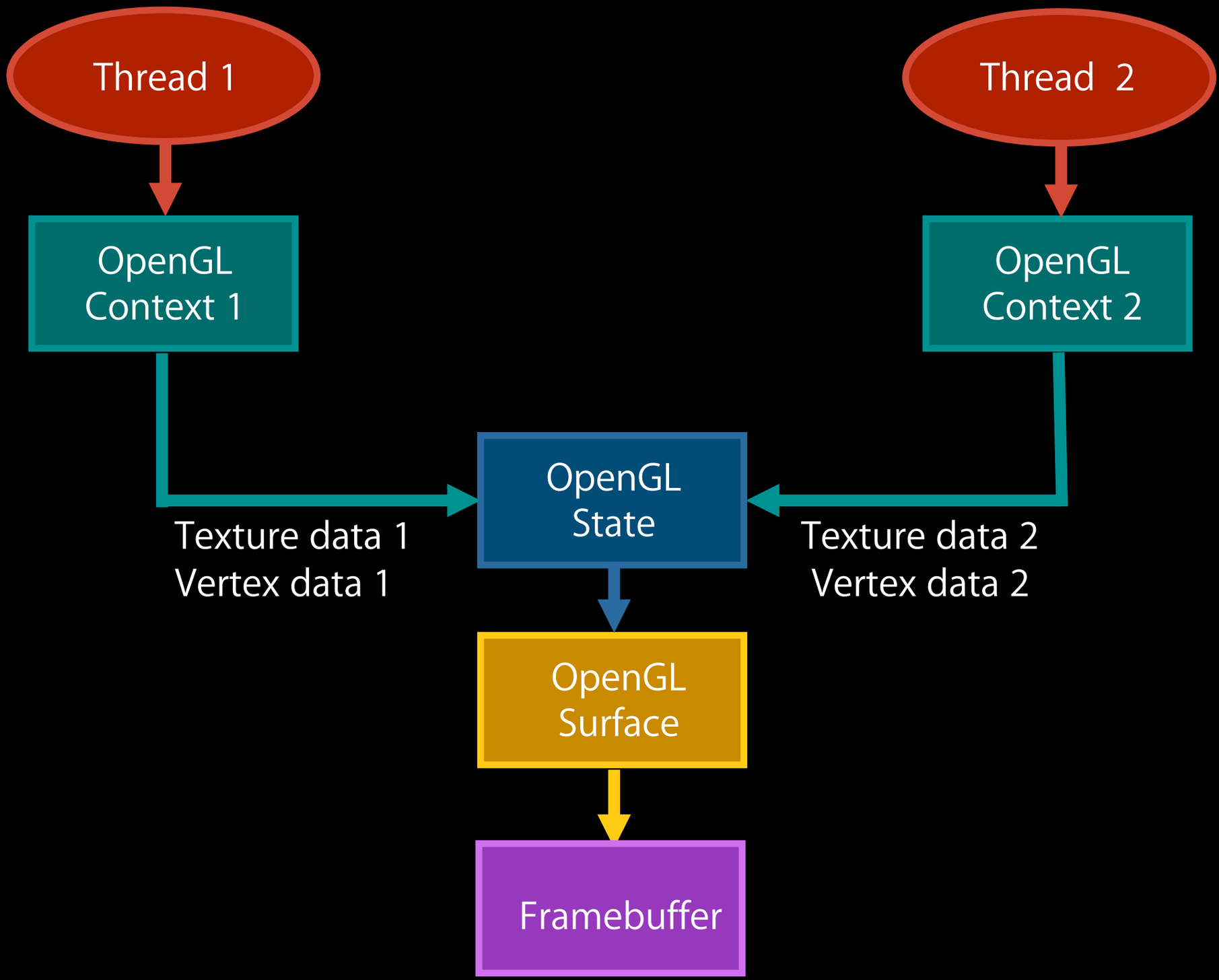
Split OpenGL Surface Processing



Split Frame Buffer Processing



Split Frame Buffer Processing



Shared OpenGL Context State/Surface

- Share context and surface sample

```
NSOpenGLContext *ctx;

ctx = [NSOpenGLContext alloc];

[ctx initWithFormat:[self pixelFomat] sharedContext:
   [self openglContext]];

[ctx makeCurrentContext];

[ctx setView:[[self openglContext] view]];
```

Shared OpenGL Context State

- Share context state sample

```
NSOpenGLContext *ctx;

ctx = [NSOpenGLContext alloc];

[ctx initWithFormat:[self pixelFomat] sharedContext:
   nil];

[ctx makeCurrentContext];

[ctx setView:[[self openglContext] view]];
```

Thread Synchronization

- Standard OS thread locking
  - NSThread
  - NSLock
- GL_APPLE_fence
  - Insert tokens into the OpenGL command stream
  - Test for token completion
  - Test for object completion

    GL_TEXTURE, GL_VERTEX_ARRAY

Using Threads

- GL_APPLE_fence sudo code

```
(draw)
glSetFenceAPPLE(name);

/* Wait for drawing to complete */
glFinishFenceAPPLE(name);

-------------------------------------------------------

(draw using VertexArrayObject or TextureObject)

/* Wait for drawing to complete */
glFinishObjectAPPLE(<GL_TEXTURE, GL_VERTEX_ARRAY>, name);
```

Demo

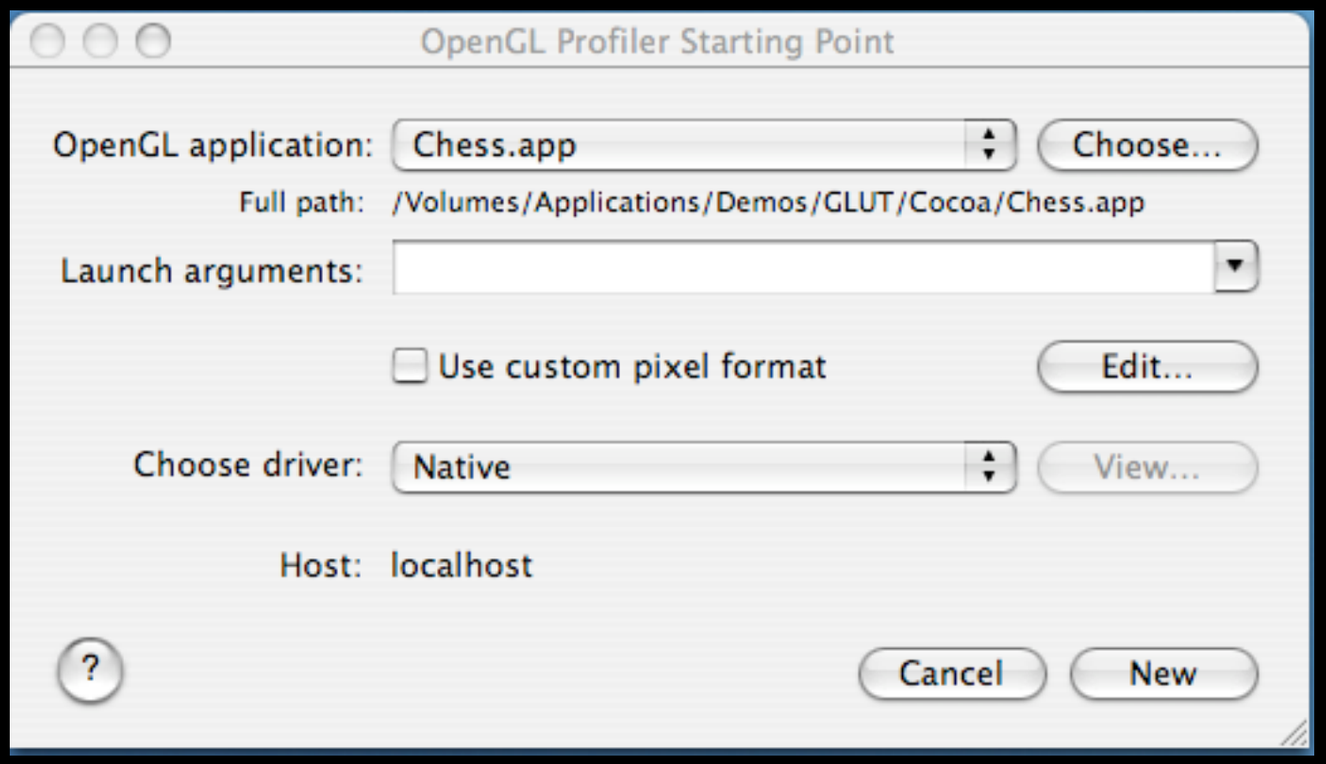
Vertex Optimization Sample Code

Using the OpenGL Profiler

- Optimizing
- Debugging
- Experimenting

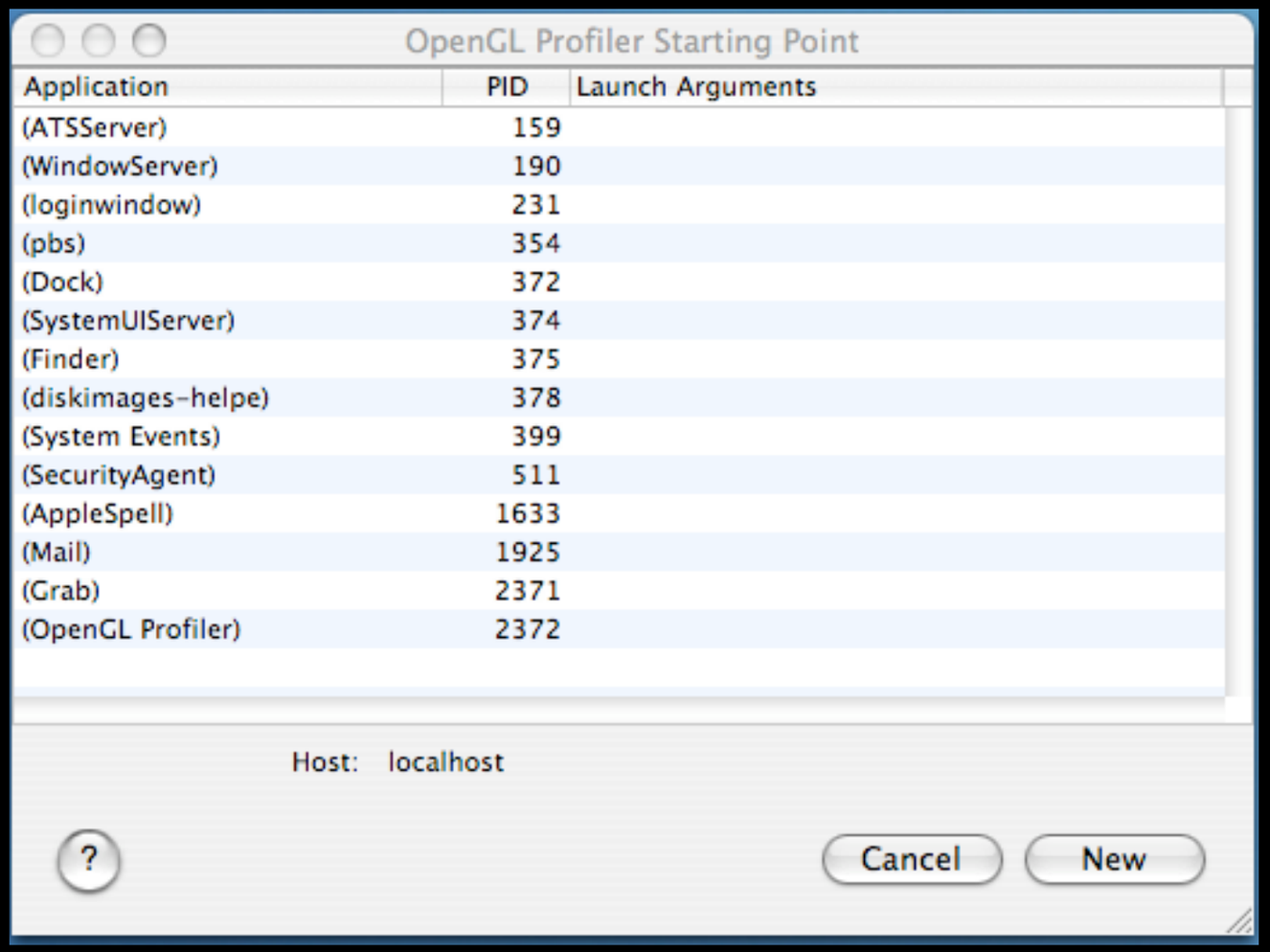
How To Use the *OpenGL Profiler*

- Launch OpenGL applications



How To Use the *OpenGL Profiler*

- Attach to OpenGL applications



How To Use the *OpenGL Profiler*

- Function statistics

Untitled: OpenGL Function Statistics

GL Function	# of Calls	Total Time	Avg Time (µsec)	% GL Time	% App Time ▲
glCallList	28,076	3746316	133.43	95.06	78.85
glBegin	2,629	41108	15.64	1.04	0.87
glVertex3f	134,908	35632	0.26	0.90	0.75
glClear	439	25828	58.84	0.66	0.54
glRotatef	13,284	15595	1.17	0.40	0.33
glTranslatef	38,682	12199	0.32	0.31	0.26
glScalef	28,514	10643	0.37	0.27	0.22
glPopMatrix	30,705	8912	0.29	0.23	0.19
glTexCoord2f	133,152	8874	0.07	0.23	0.19
glEnd	2,629	6502	2.47	0.16	0.14
glPushMatrix	30,706	5783	0.19	0.15	0.12
glMaterialfv	6,579	5498	0.84	0.14	0.12
glLightfv	1,316	3817	2.90	0.10	0.08
glDisable	3,945	3718	0.94	0.09	0.08
glBindTexture	1,314	2172	1.65	0.06	0.05
glEnable	3,946	1934	0.49	0.05	0.04
glMaterialf	2,193	1884	0.86	0.05	0.04
glMultMatrixd	439	1065	2.43	0.03	0.02
glClipPlane	877	938	1.07	0.02	0.02
glLoadIdentity	439	524	1.20	0.01	0.01
glTranslated	439	378	0.86	0.01	0.01

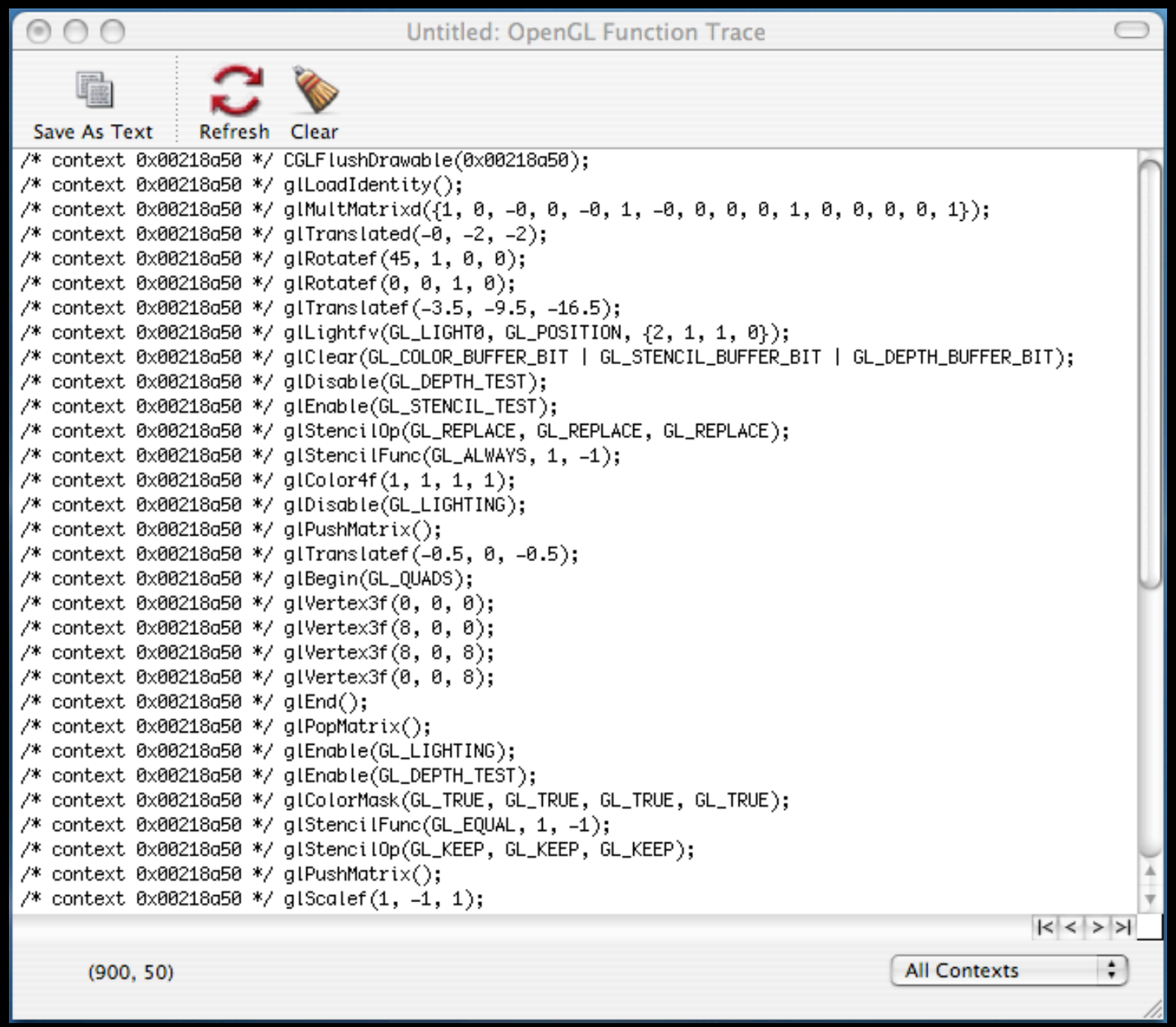
Total elapsed GL function time: 3940967.07 µsec

Estimated % time in GL: 82.94%

Show slice: < 19 of 19 > Context ID: All Contexts

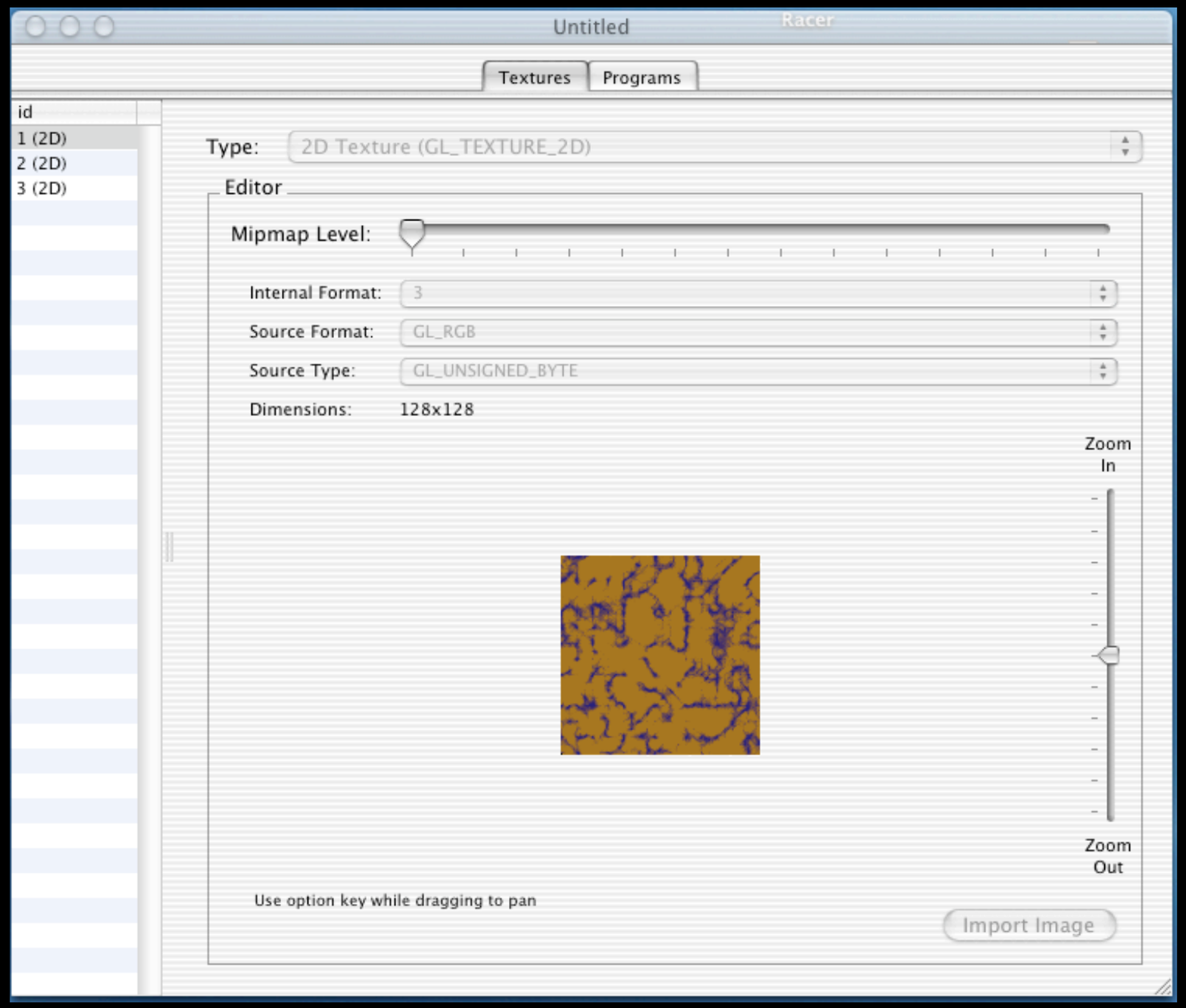
How To Use the *OpenGL Profiler*

- Function call trace



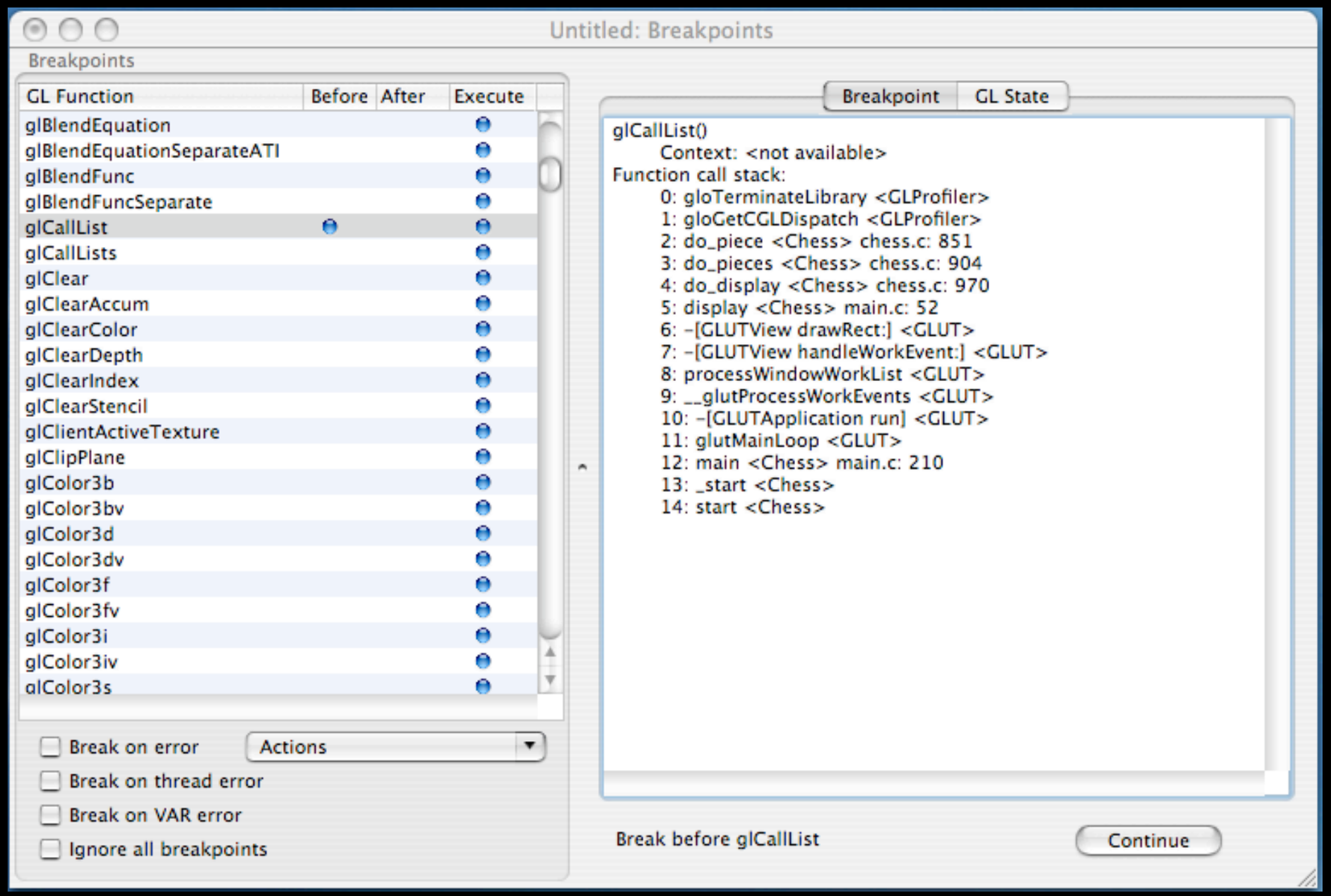
How To Use the *OpenGL Profiler*

- Textures, vertex and pixel programs



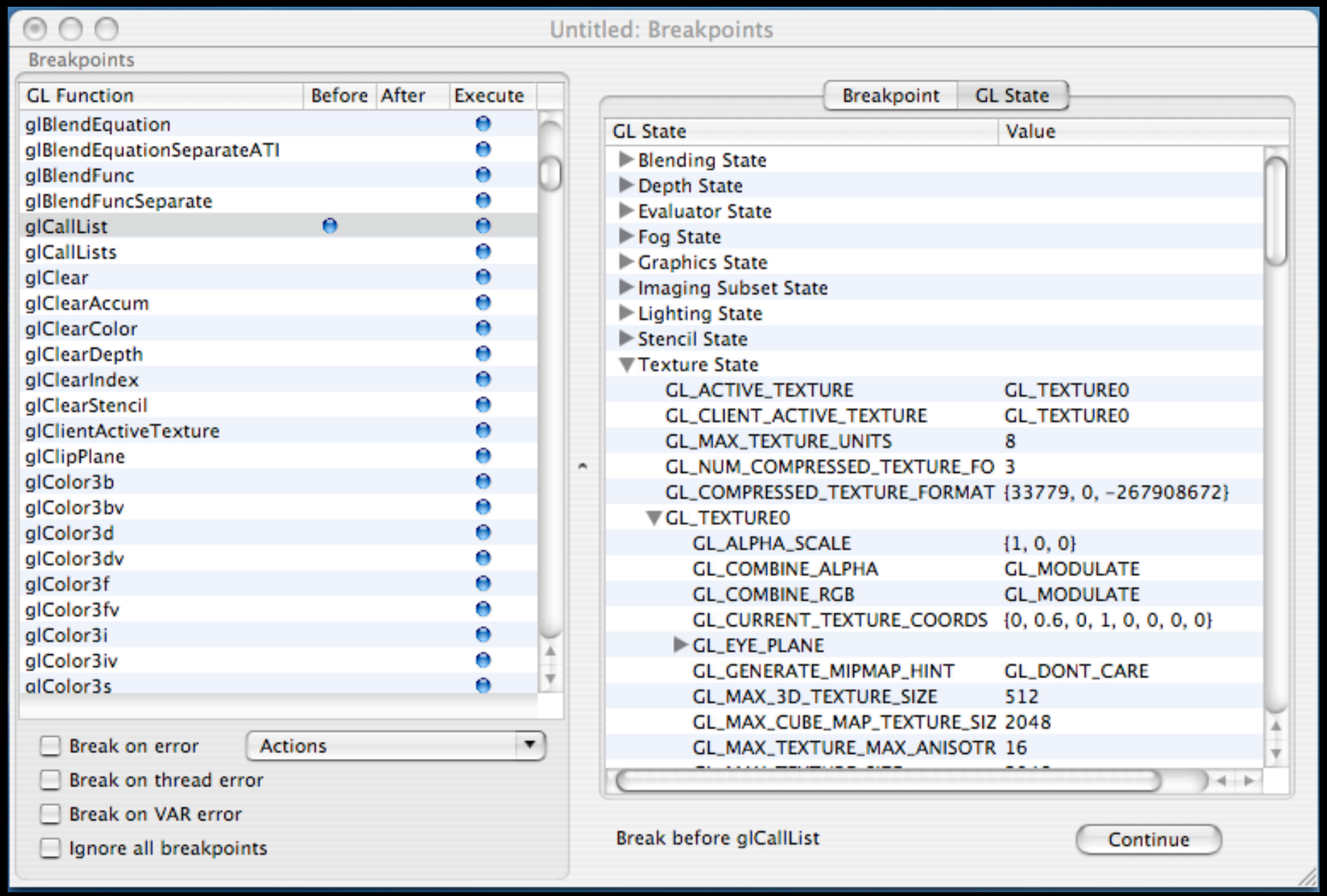
How To Use the *OpenGL Profiler*

- Set break points
  - Application call stack



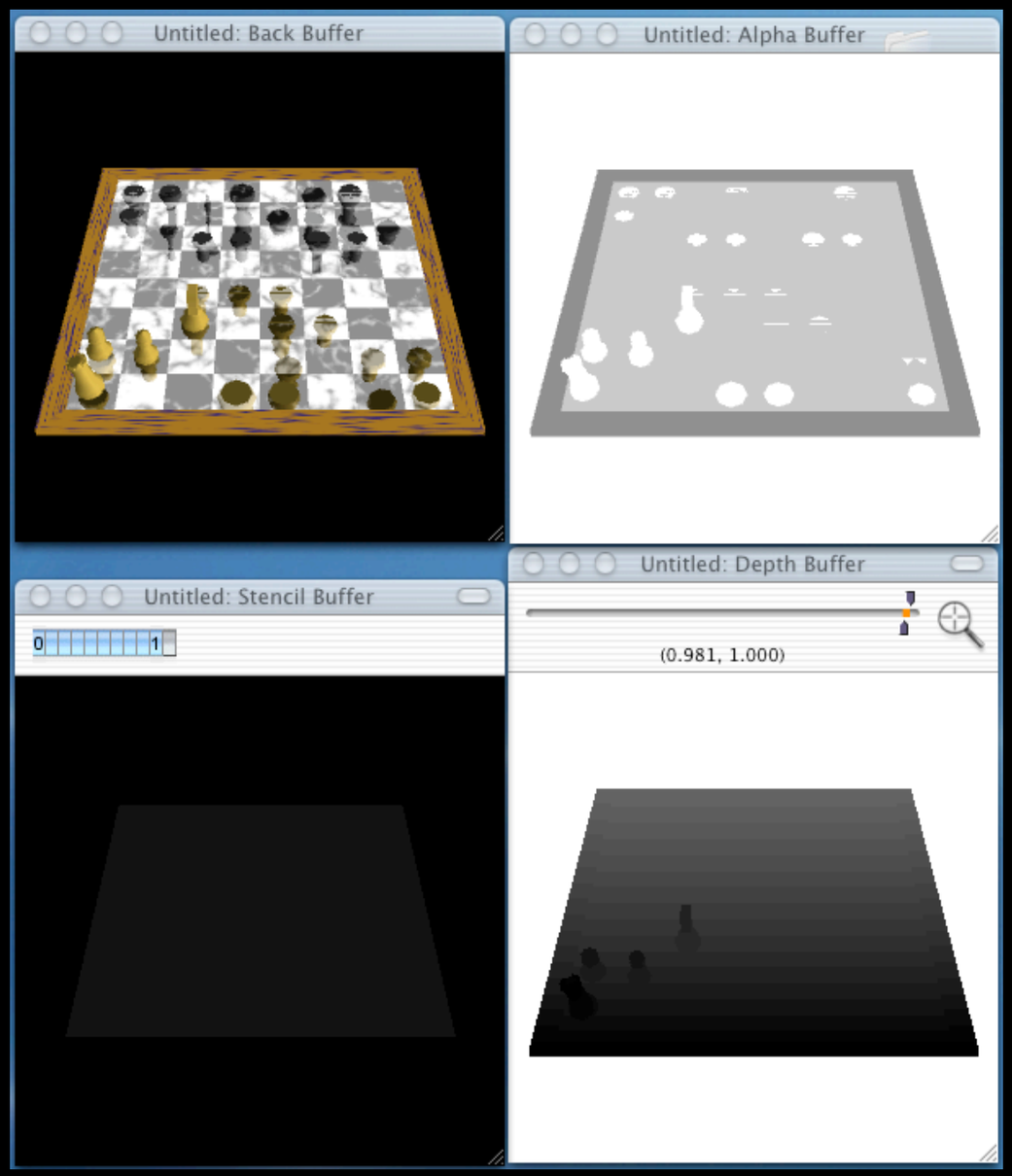
How To Use the *OpenGL Profiler*

- Set break points
  - State



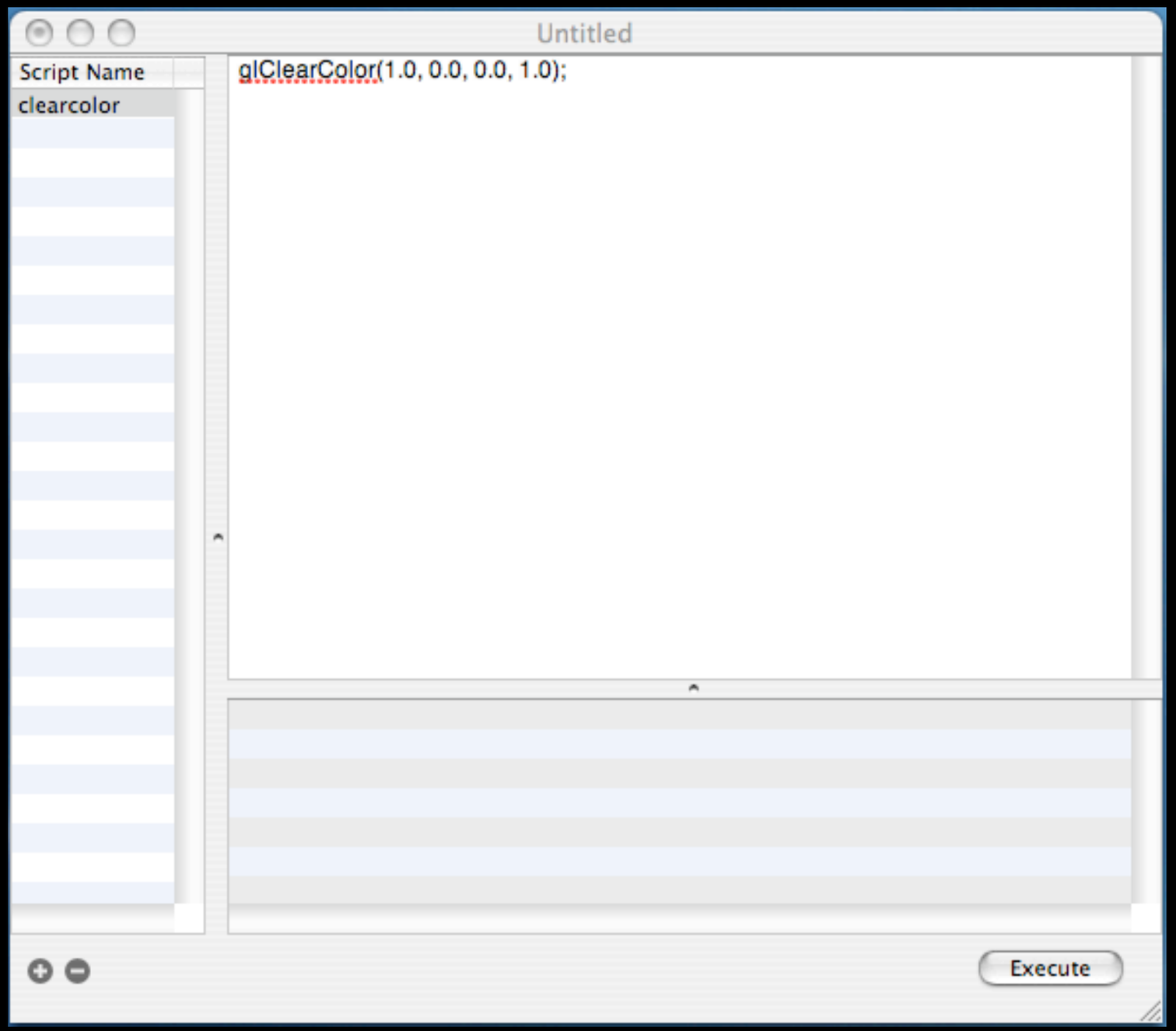
How To Use the *OpenGL Profiler*

- Offscreen buffers



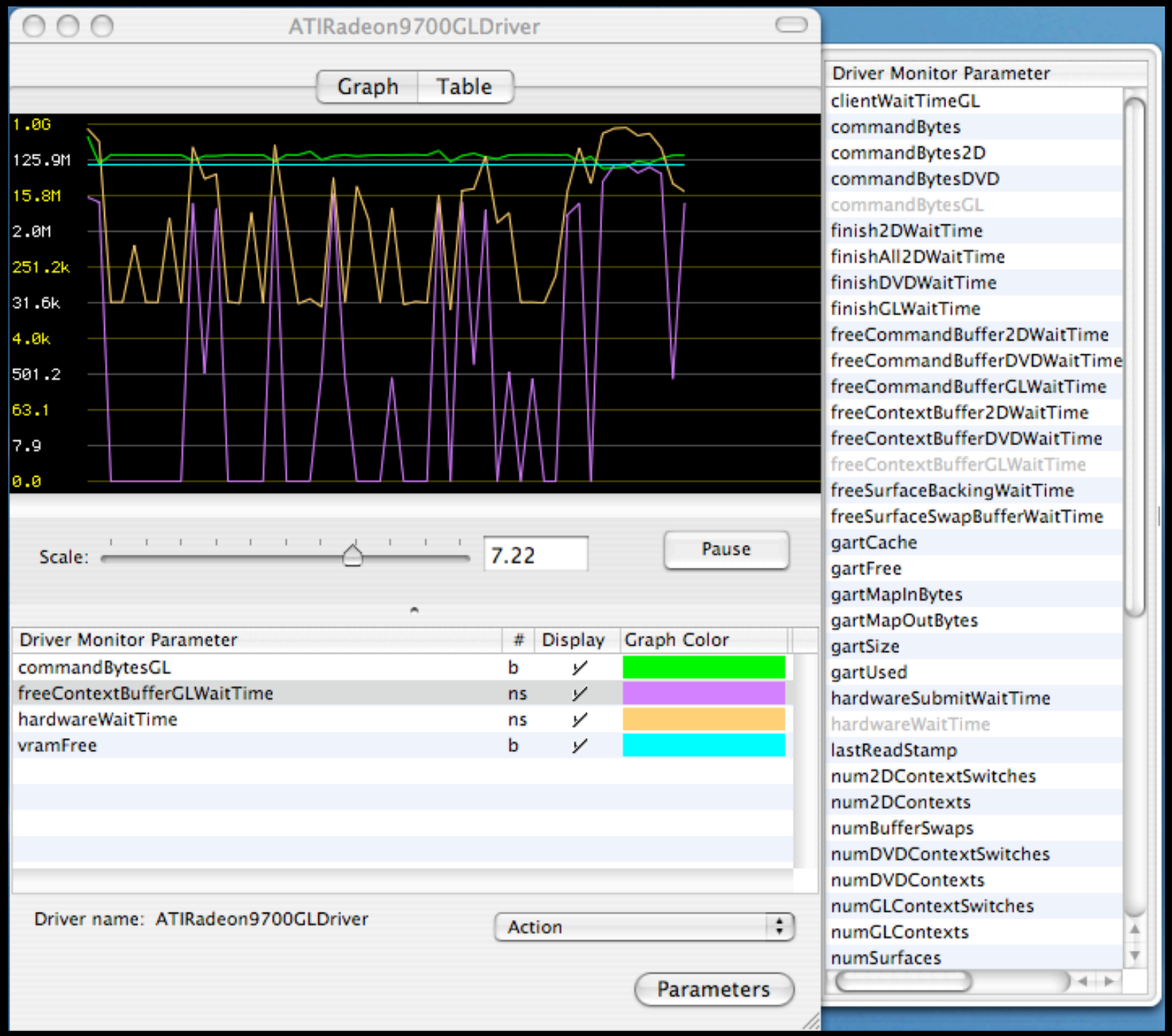
How To Use the *OpenGL Profiler*

- Command scripts



How To Use the *OpenGL Profiler*

- OpenGL Driver Monitor



Demo

OpenGL Profiler

Wrap Up

Wrap Up

- Texture Optimizations
  - Minimize CPU copies of pixel data
  - Power-of-two and non power-of-two
- Vertex Optimizations
  - Use GL_APPLE_vertex_array_range for dynamic data
    GL_STORAGE_SHARED_APPLE storage hint
  - Cache static data in VRAM
    - Use GL_APPLE_vertex_array_range with
      GL_STORAGE_CACHED_APPLE storage hint
    - Display lists
  - Offload CPU work to GPU with Vertex Programs

Wrap Up

- Use threads
  - Share OpenGL context state and surfaces
- Using glDrawPixels for small one shot images
- Using glCopyPixels fast VRAM to VRAM copying
- OpenGL Profiler
  - Find hot spots in OpenGL
  - Find stalls in OpenGL driver

Who to Contact

John Stauffer
OpenGL Engineering Manager
stauffer@apple.com

Travis Brown
Graphics & Imaging Evanglist
travis@apple.com

http://developer.apple.com/wwdc2003/urls.html

For More Information

Reference Library

Web Sites

Apple OpenGL Developer Site
http://developer.apple.com/opengl

OpenGL.org Site
http://www.opengl.org

Reference Library

Documentation

OpenGL
CGL
NSGL
AGL

See documentation link on Apple OpenGL developer site

Roadmap

200 Graphics and Imaging Overview Latest developments in Mac OS X graphics APIs.	**Presidio** **Tue., 10:30am**
201 Mac OS X OpenGL in Depth Introduction to OpenGL on Mac OS X.	**Nob Hill** **Tue., 2:00pm**
202 Technology Magic of Keynote How Keynote uses Mac OS X graphics technologies.	**Nob Hill** **Tue., 3:30pm**
204 Image Capture Update Integrating Image Capture with your product.	**Marina** **Wed., 9:00am**

Roadmap

205 Vertex Programming with OpenGL Super-fast geometry transformations using the GPU.	**Marina** **Wed., 3:30pm**
206 ColorSync Update Latest developments in Mac OS X color management.	**Mission** **Wed., 5:00pm**
207 Quartz 2D in Depth Harnessing the power of Quartz 2D in your product.	**Presidio** **Thu., 9:00am**
208 Fragment Programming with OpenGL Creating incredible per-pixel effects on Mac OS X.	**Pacific Heights** **Thu., 10:30am**

Roadmap

209 OpenGL Optimizations Obtaining maximum 3D performance on Mac OS X.	**Marina** **Thu., 2:00pm**
210 Mac OS X Printing Update Latest developments in the Mac OS X printing system.	**Mission** **Thu., 3:30pm**
211 Introduction to Quartz Services Capturing & configuring displays on Mac OS X.	**Pacific Heights** **Fri., 9:00am**
212 Cutting-Edge OpenGL Techniques Expert recipes for stunning 3D effects.	**Russian Hill** **Fri., 2:00pm**

Roadmap

421 Mac OS X Accessibility Latest developments in Mac OS X accessibility.	**Presidio** **Fri., 10:30am**
FF018 Graphics & Imaging Feedback Share your thoughts on graphics & imaging technologies.	**North Beach** **Fri., 5:00pm**

Q&A

John Stauffer
OpenGL Engineering Manager
stauffer@apple.com

http://developer.apple.com/wwdc2003/urls.html

WWDC 2003

Worldwide Developers Conference